APPENDIX A

```
Smalltalk defineClass: #AbstractParticipantProcess
	superclass: #{Core.Object}
	indexedType: #none
	private: false
	instanceVariableNames: 'myId '
	classInstanceVariableNames: ''
	imports: ''
	category: ''!


!AbstractParticipantProcess methodsFor: 'accessing'!
host

	^SocketAccessor getHostname!
myId

	^myId!
!


!AbstractParticipantProcess methodsFor: 'ui support'!
statusString

	self subclassResponsibility.!
!

Smalltalk defineClass: #ParticipantProcess
	superclass: #{AbstractParticipantProcess}
	indexedType: #none
	private: false
	instanceVariableNames: 'usp executionProcess outboundChannels outputBuffers session '
	classInstanceVariableNames: ''
	imports: ''
	category: ''!


!ParticipantProcess methodsFor: 'accessing - log'!
at: indent log: aString

	USPControllerService current at: indent log: aString.!
!


!ParticipantProcess methodsFor: 'private-execution'!
deleteJobsFromBuffer: outBuff forChannel: outChan

	[outChan isEmpty] whileFalse: [
		| deletionCount |
		deletionCount := outBuff deleteJobsUpTo: outChan next.
		self at: 0 log:
			self myId printString, ': deleted ', deletionCount printString,
			' jobs previously sent to #', outChan remoteId printString].!
executionLoop
	"Execute me forever (or until stopped by another process). This runs in a background process,
	 so use my session to allow use of the miosoft user interface while it's running."

	self subclassResponsibility.!
!


!ParticipantProcess methodsFor: 'initialize-release'!
initializeForIndex: anInteger
	"Does not need to be called within a transaction."

	myId := anInteger.

	session := OoSession open: 'currentdb'.
	session rpcTimeout: 5000; useIndex: true; recover: false.

	session transactionMROW: [
		self initializeWithinTransaction].!
!
```

```
!ParticipantProcess methodsFor: 'private-initialization'!
initializeWithinTransaction
	"Only called within a transaction (of my session)."

	usp := UpdateStreamProcessor named: 'Main' inSession: session.
	outboundChannels := Array new: usp numberOfContentionSpaces.

	self setupOutboundChannels.!
!


!ParticipantProcess methodsFor: 'testing'!
isLocal

	^true!
!


!ParticipantProcess methodsFor: 'accessing'!
numberOfContentionSpaces

	^outboundChannels size!
!


!ParticipantProcess methodsFor: 'private-outbound'!
setupOutboundChannels
	"Must be in transaction, preferrably MROW."

	1 to: outboundChannels size do: [:i |
		| channel |
		channel := JobChannel new.
		channel myId: myId.
		channel socket: nil.
		channel remoteId: i.
		outboundChannels at: i put: channel].!
!


!ParticipantProcess methodsFor: 'controlling'!
start

	executionProcess notNil ifTrue: [^self].
	executionProcess := Process
		forBlock: [self executionLoop]
		priority: Processor userSchedulingPriority - 3.  "user interaction takes precedence."
	executionProcess resume.!
stop

	| p |
	(p := executionProcess) notNil ifTrue: [p terminate].
	executionProcess := nil.
	outboundChannels do: [:ch | ch notNil ifTrue: [ch stop]].!
!

-------------------------------------------------------------------------------

Smalltalk defineClass: #ContentionSpaceProcess
	superclass: #{ParticipantProcess}
	indexedType: #none
	private: false
	instanceVariableNames: 'contentionSpace listenerProcess listenProtect inboundChannels readyJobs runningJob runningTag groupsByTag indexedWaiters
currentGroup createdJobList pleaseCommitTimer memento lastGC idleDeletionDelay previousInbounds affectedInbounds '
	classInstanceVariableNames: ''
	imports: ''
	category: ''!
```

```
!ContentionSpaceProcess methodsFor: 'accessing'!
addJob: aJob
	"This is the entry point used by jobs already executing within my executionLoop.  It allows them to
	 add new jobs safely and efficiently."

"	aJob checkContentionSpace." "safety"
	aJob ooIsPersistent ifTrue: [
		self error: 'Jobs to be transmitted must not be persistent'].
	aJob originatorIndex: myId.
	createdJobList add: aJob.!
!


!ContentionSpaceProcess methodsFor: 'private-execution'!
addWaitingJob: aJob
	"Given a job, add it to the parallel ordered collections indexedWaiters (transient) and waitingSyncJobs
	 (persistent).  Do this by tacking the job onto the end of these collections.  Answer the index->job pair
	 that was added to indexedWaiters."

	| index assoc |
	aJob assertType: Job.
	index := indexedWaiters size + 1.
	[index = (contentionSpace waitingSyncJobs size + 1)] assert.
	assoc := index -> aJob.
	indexedWaiters addLast: assoc.
	contentionSpace waitingSyncJobs addLast: aJob.
	^assoc!
collapseIfNecessary: indexedJobs
	"Given a collection of index->Job pairs, ask each job if it wants to collapse.  If the job answers nil, ask the
	 next job.  If all jobs answer nil, return the original collection indexedJobs.  Otherwise one of the jobs has
	 answered a new job to use in place of the collection.  Delete the given jobs and insert the new job (which
	 must have the same synchronization tag as all given jobs) into the indexedWaiters and waitingSyncJobs
	 collections.  Answer a collection with just newIndex->newJob in it in that case."

	| jobs |
	jobs := indexedJobs collect: [:assoc | assoc value].
	jobs do: [:job |
		| collapsed collapsedAssoc |
		collapsed := job collapseJobs: jobs.
		collapsed notNil ifTrue: [
			"A collapse has been requested by one of the jobs."
			collapsed assertType: Job.
			[jobs all: [:j | j ~~ collapsed]] assert.
			collapsed
				originatorIndex: 0;
				jobId: 0;
				tag: job tag.

			"Remove the old jobs first..."
			indexedJobs do: [:jobAssoc |
				self removeWaitingJobAssoc: jobAssoc.
				jobAssoc value ooIsPersistent ifTrue: [
					jobAssoc value ooDelete]].
			collapsedAssoc := self addWaitingJob: collapsed.
			^OrderedCollection with: collapsedAssoc]].

	"All the jobs agree that collapsing is not appropriate."
	^indexedJobs!
!


!ContentionSpaceProcess methodsFor: 'execution tuning'!
deletionRequestGranularity
	"Answer approximately how many jobs should be executed (from one source) before
	 bothering to tell the source that it may now delete them."

	^1000 "jobs"!
!
```

```
!ContentionSpaceProcess methodsFor: 'private-execution'!
executionLoop
	"Execute me forever (or until stopped by another process).  This runs in a background process,
	 so use my private session to allow use of the miosoft user interface while it's running."

	"Reconstruct the transient data structures from the persistent data."
	self recover.

	lastGC := Time millisecondClockValue.
	[ "repeat"
		| starts stops steps lastCommit anyJobsReady jobsToSend indexOrder |
		starts := stops := steps := 0.
		pleaseCommitTimer := Delay forMilliseconds: self targetTransactionTime.
		pleaseCommitTimer startup.
		lastCommit := Time millisecondClockValue.
		ObjectMemory quickGC.
		session transactionMROW: [
			[
				createdJobList := OrderedCollection new.

				"Process the jobs that are ready."
				readyJobs size > 0 ifTrue: [
					self at: 0 log: ' (', readyJobs size printString, ' jobs ready, ',
						indexedWaiters size printString, ' jobs waiting to sync)'].
				[
					"Ensure we have a (started) job in runningJob, grabbing (and starting) one if necessary from the
					 currentGroup, or if that is empty then from readyJobs."
					(runningJob == nil and: [currentGroup isEmpty]) ifTrue: [
						runningTag := nil].
					(runningJob == nil and: [currentGroup notEmpty]) ifTrue: [
						"Grab jobs from currentGroup first.  This is a synchronization group that was activated
						 when the final synchronized job of a group was consumed from readyJobs."
						| jobAssoc |
						jobAssoc := currentGroup removeFirst.
						runningJob := jobAssoc value.
						starts := starts + 1.
						memento := runningJob start.
						self removeWaitingJobAssoc: jobAssoc.
						[runningTag = runningJob tag] assert].
					[runningJob == nil and: [readyJobs notEmpty]] whileTrue: [
						| delta |
						[currentGroup isEmpty] assert.
						runningJob := readyJobs removeFirst.
						delta := runningJob jobId - (self lastJobExecutedFrom: runningJob originatorIndex) bitAnd: 16rFFFFFFFF.
						delta = 1
							ifTrue: [
								runningTag isNil
									ifTrue: [
										"A job from the readyJobs list."
										runningJob tag isNil
											ifTrue: [
												"A non-synchronized job from the readyJobs list."
												starts := starts + 1.
												memento := runningJob start]
											ifFalse: [

												"A synchronized job from the readyJobs list."
												| assoc pair |
												assoc := self addWaitingJob: runningJob.
												pair := groupsByTag at: runningJob tag ifAbsentPut: [0 -> OrderedCollection new].
												pair key: pair key + runningJob quorumFraction.
												pair value add: assoc.
												self registerAsConsumed: runningJob.
												pair key = 1
													ifTrue: [
														"The last job of a synchronized group just arrived.  Start executing the group."
														pair value: (self collapseIfNecessary: pair value).
														currentGroup := pair value.
														groupsByTag removeKey: assoc value tag.
														assoc := currentGroup removeFirst.
														self removeWaitingJobAssoc: assoc.
														runningJob := assoc value.
														runningTag := runningJob tag.
														starts := starts + 1.
														memento := runningJob start]
```

```
                                                    ifFalse: [
                                                        "Wait for the other synchronizing jobs to show up."
                                                        runningJob := nil]]]
                                            ifFalse: [
                                                "The job was taken from currentGroup, so just start executing it."
                                                starts := starts + 1.
                                                memento := runningJob start]]
                                    ifFalse: [
                                        [delta = 0 or: [delta > 16r80000000]] assert: 'Bug -- job is out of order'.
                                        "This was a retransmitted job (due to a disconnect/restart).  We must ignore it."
                                        runningJob := nil].
                            ].
                            pleaseCommitTimer inProgress not or: [runningJob == nil]
                        ] whileFalse: [
                            "Run this job to completion, or until a commit is desired."
                            [
                                pleaseCommitTimer inProgress not or: [runningJob atEnd: memento]
                            ] whileFalse: [
                                steps := steps + 1.
                                memento := runningJob step: memento withScheduler: self.
                            ].
                            (runningJob atEnd: memento) ifTrue: [
                                runningJob finish: memento withScheduler: self.
                                stops := stops + 1.
                                memento := 'bogosity'.  "nobody will see this value"
                                runningJob tag isNil ifTrue: [
                                    "Tagged jobs were already registered as consumed when they were taken from the
                                     readyJobs list, examined, and added to the (persistent) synchronized job data structures."
                                    self registerAsConsumed: runningJob].
                                runningJob ooIsPersistent ifTrue: [runningJob ooDelete].
                                runningJob := nil].
                        ]. "while there are jobs"
                        "Update the persistent objects in preparation for the upcoming commit."
                        createdJobList do: [:job |
                            (outputBuffers at: job contentionIndex) addJob: job].
                        jobsToSend := outputBuffers collect: [:buff |
                            buff prepareToCommit].
                        runningTag := runningJob isNil ifTrue: [nil] ifFalse: [runningJob tag].
                        contentionSpace runningJob: runningJob.
                        contentionSpace runningTag: runningTag.
                        self at: 0 log: '[committing {',
                            starts printString, ' starts, ',
                            steps printString, ' steps, ',
                            stops printString, ' stops  = ',
                            (starts + stops / 2.0) printString, ' jobs @ ',
                            (((starts + stops / 2.0) / (Time millisecondClockValue - lastCommit) * 10000.0) rounded / 10.0) printString, ' jobs/s}...'.
                        anyJobsReady := runningJob notNil or: [currentGroup size > 0 or: [readyJobs size > 0]].
                    ] valueOnUnwindDo: [session succeed: false].
                ]. "end transaction"
                self at: 1 log: 'done, sending ', (jobsToSend collect: [:x | x size]) sum printString, ' jobs...'.
                pleaseCommitTimer disable.

                indexOrder := (1 to: jobsToSend size) asSortedCollection: [:i1 :i2 |
                    "Simple trick, but it tends to randomize the order that output channels get processed."
                    (jobsToSend at: i1) size > (jobsToSend at: i2) size].
                indexOrder do: [:i |
                    (outboundChannels at: i) nextPutAll: (jobsToSend at: i)].
                self at: 1 log: 'done, sending replies...'.
                self transmitDeletionsWithGranularity: self deletionRequestGranularity.
                self at: 1 log: 'done]'.

                (Time millisecondClockValue between: lastGC and: lastGC + self forcedGarbageCollectFrequency) ifFalse: [
                    "Force garbage collect every few minutes."
                    ObjectMemory garbageCollect.
                    lastGC := Time millisecondClockValue].

                self waitForJobsAndDoMaintenance.

            ] repeat.!
!
```

```
!ContentionSpaceProcess methodsFor: 'execution tuning'!
forcedGarbageCollectFrequency
	"Answer the approximate number of milliseconds we should wait before forcing a
	garbage collection.  We only force these garbage collects because (1) the incremental
	collector depends on user activity, and (2) the generational scavenger tenures pretty
	much every job that we receive, long before we have a change to execute it (and then
	let it go).  These forced GC's allow our growth regime bound to be set fairly high (based
	on actual ram) without expecting to actually reach it in normal use."

	^120000 "ms, = 2 minutes"!
!


!ContentionSpaceProcess methodsFor: 'private-initialization'!
initializeWithinTransaction
	"Only called within a transaction (of my session)."

	super initializeWithinTransaction.
	listenProtect := Semaphore forMutualExclusion.
	contentionSpace := usp participantAt: myId.
	inboundChannels := Array new: usp numberOfParticipants.

	self setupListenerProcess.!
!


!ContentionSpaceProcess methodsFor: 'private-execution'!
lastJobExecutedFrom: originatorIndex
	"Answer the id of the job from the given contention space that has most recently executed."

	^contentionSpace lastJobsExecuted at: originatorIndex!
lastJobExecutedFrom: originatorIndex is: anInteger
	"Set the most recent job id from the given contention space that has executed."

	contentionSpace lastJobsExecuted at: originatorIndex put: anInteger!
!
```

```
!ContentionSpaceProcess methodsFor: 'private-inbound'!
listenerLoopForSocket: socket

    [
        [
            | conn channel newId |
            self at: 3 log: 'Waiting for connection...'.
            socket readWait.
            self at: 4 log: 'connection is present'.
            conn := socket acceptNonBlock.
            conn isNil
                ifTrue: [
                    self at: 4 log: 'false alarm - not connected'.
                    (Delay forSeconds: 2) wait]
                ifFalse: [
                    self at: 4 log: 'connection is from host ', conn getPeer hostName.
                    channel := JobChannel new.
                    channel myId: myId.
                    channel remoteId: 'Unknown'.
                    channel socket: conn.
                    channel onConnect: [:chan |
                        | oldChannel |
                        [chan == channel] assert.
                        self at: 4 log: 'connection from ', chan remoteId printString, ' is established'.
                        newId := chan remoteId.
                        [newId isInteger] assert.
                        listenProtect critical: [
                            oldChannel := inboundChannels at: newId.
                            inboundChannels at: newId put: chan].
                        oldChannel notNil ifTrue: [oldChannel stop]].
                    channel onDisconnect: [:chan |
                        self at: 4 log: 'Lost connection from ', newId printString.
                        chan stop.
                        listenProtect critical: [
                            (inboundChannels at: newId) == chan ifTrue: [
                                inboundChannels at: newId put: nil]]].
                    channel start].
        ] repeat.
    ] ensure: [
        socket close
    ].!
!


!ContentionSpaceProcess methodsFor: 'execution tuning'!
maximumIdleDeletionDelay
    "Answer the maximum time to wait from when a deletion request arrives to when the
     deletion should actually be committed.  This only applies when there are no jobs left
     to execute.  As soon as a job arrives (assuming I was previously quiescent), we just
     commit the deletion requests right away and start executing."

    ^10000 "ms"!
!


!ContentionSpaceProcess methodsFor: 'accessing'!
nextUniqueInteger
    "This should only be called by jobs running in my executionLoop (and therefore in a non-conflicting transaction)."

    ^contentionSpace nextUniqueInteger!
!
```

```
!ContentionSpaceProcess methodsFor: 'private-execution'!
recover
    "Reconstruct the transient data structures from the persistent data."

    IndexEntry initialize.

    session transactionMROW: [
        outputBuffers := contentionSpace outputBuffers copy.
        "Start up the listener if it was stopped..."
        self setupListenerProcess.

        "Recovery -- resend all outbound jobs.  First process the deletion requests to reduce the
         amount of useless information we're resending."
        outboundChannels with: outputBuffers do: [:outChan :outBuff |
            outChan reestablishIfNecessaryFor: usp.
            self deleteJobsFromBuffer: outBuff forChannel: outChan.
            outChan wasRecovered ifTrue: [
                outBuff retransmitAllViaChannel: outChan]].
        readyJobs := OrderedCollection new.
        runningJob := contentionSpace runningJob.
        runningTag := contentionSpace runningTag.
        currentGroup := OrderedCollection new.
        groupsByTag := Dictionary new.
        indexedWaiters := OrderedCollection new.
        contentionSpace waitingSyncJobs keysAndValuesDo: [:i :job |
            | assoc pair |
            [job ~~ runningJob] assert.
            assoc := i -> job.
            job tag = runningTag ifTrue: [
                currentGroup add: assoc].
            indexedWaiters addLast: assoc.
            [indexedWaiters size = i] assert.
            pair := groupsByTag at: job tag ifAbsentPut: [0 -> OrderedCollection new].
            pair key: pair key + job quorumFraction.
            [pair key < 1] assert.
            pair value add: assoc].
        runningJob == nil
            ifTrue: [memento := nil]
            ifFalse: [memento := runningJob restart].
        previousInbounds := (1 to: inboundChannels size) collect: [:i |
            self lastJobExecutedFrom: i].
        affectedInbounds := previousInbounds copy.
    ].!
registerAsConsumed: aJob
    "Record the fact that this job has been 'completed'.  For a non-synchronizing job, this means the
    job really has finished executing.  For a synchronizing job it means the job's essence has been
    recorded persistently locally (and is therefore recoverable).  Note that we can't immediately send
    a message back to the originator, but instead must wait until we've committed this job's effect."

    [(aJob jobId - (self lastJobExecutedFrom: aJob originatorIndex) bitAnd: 16rFFFFFFFF) = 1] assert.
    self lastJobExecutedFrom: aJob originatorIndex is: aJob jobId.
    affectedInbounds at: aJob originatorIndex put: aJob jobId.
    (previousInbounds at: aJob originatorIndex) isNil ifTrue: [
        previousInbounds at: aJob originatorIndex put: aJob jobId - 1].!
removeWaitingJobAssoc: jobAssoc
    "Given an index->Job pair, remove this job from the parallel ordered collections indexedWaiters
    (transient) and waitingSyncJobs (persistent).  Do this by moving the last job into the hole that
    would be left by removing the designated job."

    | lastAssoc |
    lastAssoc := indexedWaiters last.
    [lastAssoc value == contentionSpace waitingSyncJobs last] assert.
    [jobAssoc value == (contentionSpace waitingSyncJobs at: jobAssoc key)] assert.
    indexedWaiters
        at: jobAssoc key put: lastAssoc;
        removeLast.
    contentionSpace waitingSyncJobs
        at: jobAssoc key put: lastAssoc value;
        removeLast.
    lastAssoc key: jobAssoc key.!
!
```

```
!ContentionSpaceProcess methodsFor: 'accessing'!
session

	^session!
!


!ContentionSpaceProcess methodsFor: 'private-inbound'!
setupListenerProcess
	"This must be inside a transaction.  Update the ContentionSpace's host and port information once
	 the socket is ready to connect to."

	| port socket |
	listenerProcess notNil ifTrue: [^self].

	port := contentionSpace port.
	[
		socket := SocketAccessor newTCPserverAtPort: port.
	] on: OSErrorHolder existingReferentSignal do: [:ex |
		"Scan for a free port to use."
		port := port + 1.
		port > IPSocketAddress maxPort ifTrue: [port := IPSocketAddress firstUnreservedPort].
		ex retry].
	[socket notNil] assert.

	"Let everyone know (via eventual polling) which port to connect to."
	socket listenFor: contentionSpace numberOfParticipants.
	contentionSpace setHostName: SocketAccessor getHostname port: port.
	listenerProcess := Process
		forBlock: [self listenerLoopForSocket: socket]
		priority: Processor userSchedulingPriority - 1.
	listenerProcess resume.!
!


!ContentionSpaceProcess methodsFor: 'ui support'!
statusString

	| str |
	str := WriteStream on: (String new: 64).
	listenerProcess notNil ifTrue: [str nextPutAll: 'Listening, '].
	executionProcess notNil ifTrue: [str nextPutAll: 'Running, '].
	str nextPutAll: 'IN='.
	inboundChannels do: [:in |
		str nextPutAll: (
			in isNil ifTrue: ['-'] ifFalse: [
				in hasFirmConnection ifTrue: ['+'] ifFalse: ['?']])].
	str nextPutAll: ', OUT='.
	outboundChannels do: [:out |
		str nextPutAll: (
			out isNil ifTrue: ['-'] ifFalse: [
				out hasFirmConnection ifTrue: ['+'] ifFalse: ['?']])].
	^str contents!
!


!ContentionSpaceProcess methodsFor: 'controlling'!
stop

	| p |
	(p := listenerProcess) notNil ifTrue: [p terminate].
	listenerProcess := nil.
	super stop.
	inboundChannels do: [:ch | ch notNil ifTrue: [ch stop]].
	inboundChannels := Array new: inboundChannels size.
	createdJobList := nil.!
!
```

```
!ContentionSpaceProcess methodsFor: 'execution tuning'!
targetTransactionTime
        "Answer the 'ideal' length of a transaction in milliseconds.  When executing jobs, if we notice
         this amount of time has expired, we force a commit even if there are more jobs ready to run."

        ^5000  "ms"!
!
```

```
!ContentionSpaceProcess methodsFor: 'private-execution'!
transmitDeletionsWithGranularity: granularity

    1 to: affectedInbounds size do: [:i |
        | previousLastId lastJobId |
        previousLastId := previousInbounds at: i.
        lastJobId := affectedInbounds at: i.
        (lastJobId notNil and: [lastJobId ~= previousLastId]) ifTrue: [
            | in |
            (lastJobId // granularity) = (previousLastId // granularity) ifFalse: [
                "Send a deletion request if we crossed the granularity boundary (or a rollover)
                 for the completed jobs."
                in := inboundChannels at: i.
                in notNil ifTrue: [
                    previousInbounds at: i put: lastJobId.
                    in nextPut: lastJobId]]]].!
waitForJobsAndDoMaintenance
    "Wait for jobs to arrive along my inboundChannels.  Until some arrive, process job-deletion messages
     and attempt to reestablish channels if needed."

    [ "whileTrue:"
        | pollingPause |
        pollingPause := 10.  "ms"
        session transactionMROW: [
            | hasDeleted timerExpired |
            self at: 0 log: self myId printString, ': looking for jobs etc.'.
            idleDeletionDelay := Delay forMilliseconds: self maximumIdleDeletionDelay.
            idleDeletionDelay startup.
            hasDeleted := false.
            [
                "Process all deletion requests."
                outboundChannels with: outputBuffers do: [:outChan :outBuff |
                    outChan reestablishIfNecessaryFor: usp.
                    [outChan isEmpty] whileFalse: [
                        hasDeleted ifFalse: [
                            "Capture the time at which the first delete happens."
                            idleDeletionDelay disable; startup.
                            hasDeleted := true].
                        self deleteJobsFromBuffer: outBuff forChannel: outChan]].
                "Deal with incoming jobs."
                inboundChannels do: [:inChan |
                    inChan notNil ifTrue: [
                        [inChan isEmpty] whileFalse: [
                            readyJobs add: (inChan nextUsingSession: session)]]].
                "Now check to see if any connections have been re-established (and therefore need
                 their jobs re-sent)."
                outboundChannels with: outputBuffers do: [:outChan :outBuff |
                    outChan wasRecovered ifTrue: [
                        outBuff retransmitAllViaChannel: outChan]].

                timerExpired := idleDeletionDelay inProgress not.
                timerExpired ifTrue: [
                    "It's been a while since any jobs, or even deletion requests have arrived.  Now's a
                     good time to send out a *precise* deletion request for those few jobs for which I've
                     executed them but I haven't told their originators yet."
                    self transmitDeletionsWithGranularity: 1].
                "Commit the transaction whenever either:
                    (A) there are jobs ready, or
                    (B) a deletion request was processed more than N seconds ago."
                readyJobs notEmpty or: [
                    hasDeleted and: [timerExpired]]
            ] whileFalse: [
                (Delay forMilliseconds: pollingPause) wait.
                pollingPause := pollingPause * 2 min: 1000.  "Back off to leave some CPU for other OS processes."
            ].
        ].
        readyJobs isEmpty
    ] whileTrue: [].!
!
```

```
Smalltalk defineClass: #ProducerProcess
	superclass: #{ParticipantProcess}
	indexedType: #none
	private: false
	instanceVariableNames: 'sessionLock jobQueue '
	classInstanceVariableNames: ''
	imports: ''
	category: ''!


!ProducerProcess methodsFor: 'accessing'!
addJob: aJob

"	aJob checkContentionSpace." "safety"
	[aJob ooIsPersistent not] assert.
	aJob originatorIndex: myId.
	jobQueue nextPut: aJob.!
!
```

```
!ProducerProcess methodsFor: 'private-execution'!
executionLoop
	"Execute me forever (or until stopped by another process).  This runs in a background process,
	 so use my session to allow use of the miosoft user interface while it's running."

	self recover.

	[ "repeat"
		| jobsToSend allJobs check |
		allJobs := OrderedCollection new: 100.
		[
			jobQueue isEmpty
		] whileFalse: [
			allJobs add: jobQueue next.
		].
		allJobs size > 0 ifTrue: [
			self at: 0 log: 'DEBUG: allJobs size = ', allJobs size printString].
		check := outboundChannels any: [:ch | ch shouldCheck].
		check := check or: [allJobs notEmpty].
		check
			ifFalse: [
				"Be CPU-friendly when we're simply maintaining the connections."
				[allJobs isEmpty] assert.
				(Delay forSeconds: 2) wait]
			ifTrue: [
				allJobs isEmpty ifTrue: [(Delay forSeconds: 1) wait]. "Be nice if we're just trying to reconnect."
				self withSessionDo: [:s |
					s transactionMROW: [
						[
							"Process all deletion requests."
							outboundChannels with: outputBuffers do: [:outChan :outBuff |
								outChan reestablishIfNecessaryFor: usp.
								outChan wasRecovered ifTrue: [
									outBuff retransmitAllViaChannel: outChan].
								self deleteJobsFromBuffer: outBuff forChannel: outChan].
							"Update the persistent objects in preparation for the upcoming commit."
							allJobs do: [:job |
								| buff |
								[job contentionIndex between: 1 and: usp numberOfContentionSpaces] assert.
								buff := outputBuffers at: job contentionIndex.
								buff addJob: job].
							jobsToSend := outputBuffers collect: [:buff |
								buff prepareToCommit].
							self at: 0 log: 'jobsToSend = ', (jobsToSend collect: [:x | x size]) printString.
							self at: 0 log: '[commit...'.
						] valueOnUnwindDo: [s succeed: false].
					]. "end transaction"
				].
				self at: 0 log: '...done, sending jobs...'.
				jobsToSend with: outboundChannels do: [:jobList :outChan |
					jobList size > 0 ifTrue: [
						self at: 0 log: '{', jobList size printString,
							' from ', outChan myId printString,
							' to ', outChan remoteId printString, '}'.
						outChan nextPutAll: jobList]].
				self at: 0 log: '...done]'.
			]. "...ifTrue"
	] repeat.!
!


!ProducerProcess methodsFor: 'initialize-release'!
initializeForIndex: anInteger
	"Does not need to be called within a transaction."

	super initializeForIndex: anInteger.
	sessionLock := Semaphore forMutualExclusion.
	jobQueue := SharedQueue new.!
!
```

```
!ProducerProcess methodsFor: 'private-execution'!
recover
	"Reconstruct the transient data structures from the persistent data."

	IndexEntry initialize.

	"The recovery behaviour is to resend all my jobs."
	self withSessionDo: [:s |
		s transactionMROW: [
			outputBuffers := (usp participantAt: myId) outputBuffers copy.
			"Process old deletion requests (this should reduce the number of jobs needed for recovery)."
			outboundChannels with: outputBuffers do: [:outChan :outBuff |
				outChan reestablishIfNecessaryFor: usp.
				outChan wasRecovered ifTrue: [
					outBuff retransmitAllViaChannel: outChan].
				self deleteJobsFromBuffer: outBuff forChannel: outChan]]].!
!


!ProducerProcess methodsFor: 'ui suppont'!
statusString

	| str |
	str := WriteStream on: (String new: 64).
	executionProcess notNil ifTrue: [str nextPutAll: 'Running, '].
	str nextPutAll: 'OUT='.
	outboundChannels do: [:out |
		str nextPutAll: (
			out isNil ifTrue: ['-'] ifFalse: [
				out hasFirmConnection ifTrue: ['+'] ifFalse: ['?']])].
	^str contents!
!


!ProducerProcess methodsFor: 'accessing'!
withSessionDo: aBlock
	"Execute the block, passing in the session I use for executing my job-distributing loop.
	 Note that the execution process and the process calling this method will mutually
	 exclude each other from running."

	sessionLock critical: [aBlock value: session].!
!

----------------------------------------------------------------------------------------------------

Smalltalk defineClass: #SaturationTestProducerProcess
	superclass: #{ProducerProcess}
	indexedType: #none
	private: false
	instanceVariableNames: ''
	classInstanceVariableNames: ''
	imports: ''
	category: ''!


!SaturationTestProducerProcess methodsFor: 'private-outbound'!
autoReplenishJobsForSaturationTest: aBoolean
	"Must be in transaction, preferrably MROW."

	1 to: outboundChannels size do: [:i |
		| channel |
		channel := outboundChannels at: i.
		aBoolean
			ifTrue: [
				channel jobGenerator: [:n |
					"This block says how to generate more jobs via this channel."
					n timesRepeat: [
						self addJob: (RandomChangeForSaturationTestJob new
							contentionIndex: channel remoteId;
							productSubscript: (BouncingJob random next * SmallInteger maxVal) truncated + 1)]]]
			ifFalse: [channel jobGenerator: nil]].!
currentJobCounts
	"Used for saturation testing."

	^outputBuffers collect: [:ch | ch currentJobCount]!
```

```
!

!SaturationTestProducerProcess methodsFor: 'private-execution'!
deleteJobsFromBuffer: outBuff forChannel: outChan

    | deletionCount |
    deletionCount := 0.
    [outChan isEmpty] whileFalse: [
        deletionCount := deletionCount + (outBuff deleteJobsUpTo: outChan next)].

    deletionCount > 0 ifTrue: [
        self at: 0 log:
            self myId printString, ': deleted and replenishing ', deletionCount printString,
            ' jobs previously sent to #', outChan remoteId printString.

        outChan jobGenerator notNil ifTrue: [
            outChan jobGenerator value: deletionCount].
    ].!
!


!SaturationTestProducerProcess methodsFor: 'private-outbound'!
setupOutboundChannels
    "Must be in transaction, preferrably MROW."

    1 to: outboundChannels size do: [:i |
        | channel |
        channel := SaturationTestJobChannel new.
        channel myId: myId.
        channel socket: nil.
        channel remoteId: i.
        outboundChannels at: i put: channel].!
!

Smalltalk defineClass: #RemoteProcess
    superclass: #{AbstractParticipantProcess}
    indexedType: #none
    private: false
    instanceVariableNames: 'host status '
    classInstanceVariableNames: ''
    imports: ''
    category: ''!


!RemoteProcess methodsFor: 'accessing'!
host

    ^host!
!


!RemoteProcess methodsFor: 'initialize-release'!
initializeForIndex: anInteger host: hostString status: statString
    "Does not need to be called within a transaction."

    myId := anInteger.
    host := hostString.
    status := statString.!
!


!RemoteProcess methodsFor: 'testing'!
isLocal

    ^false!
!


!RemoteProcess methodsFor: 'controlling'!
start

    USPControllerService current send: 'Start process ', myId printString toHost: host.!
!
```

```
!RemoteProcess methodsFor: 'accessing'!
status: aString

	status := aString.!
!


!RemoteProcess methodsFor: 'ui support'!
statusString

	^status!
!


!RemoteProcess methodsFor: 'controlling'!
stop

	USPControllerService current send: 'Stop process ', myId printString toHost: host.!
!


Smalltalk defineClass: #Job
	superclass: #{Core.Object}
	indexedType: #none
	private: false
	instanceVariableNames: '
		contentionIndex <uint16>
		originatorIndex <uint16>
		jobId <uint32>
		tagInteger <uint64>
		quorumFractionNumerator <uint32>
		quorumFractionDenominator <uint32>'
	classInstanceVariableNames: ''
	imports: ''
	category: ''!


!Job class methodsFor: 'instance creation'!
new

	^super new initialize!
!


!Job class methodsFor: 'private: generated'!
ooCodeGenVersion

	^ 1!
ooTypedInstanceVariablesString

	^ '
		contentionIndex <uint16>
		originatorIndex <uint16>
		jobId <uint32>
		tagInteger <uint64>
		quorumFractionNumerator <uint32>
		quorumFractionDenominator <uint32>'!
!


!Job methodsFor: 'execution'!
atEnd: memento
	"Answer whether the job has finished stepping.  If so, the #finish: message will be sent to it.
	 This method is called within a transaction.  Normally, this memento was the result of the previous
	 execution step, but after a crash, the #restart message is sent to acquire a memento."

	self subclassResponsibility.!
!
```

```
!Job methodsFor: 'checking'!
checkContentionSpace
	"Make sure I'll only access objects in my own contention space.  This check can be
	 made at any time after the job has been fully initialized, including just prior to execution."

	^self!
!


!Job methodsFor: 'execution'!
collapseJobs: jobs
	"Given a collection of jobs which includes me, answer either nil or a new
	 single job that has the same effect as the given jobs.  Each job in the
	 collection will be given the chance to collapse, and the first job that replies
	 with a collapsed job will be the one that determines how to collapse them.
	 Since the jobs are persistent, this is always called within a transaction."

	^nil!
!


!Job methodsFor: 'accessing'!
contentionIndex

	^contentionIndex!
contentionIndex: anInteger

	contentionIndex := anInteger.!
!


!Job methodsFor: 'execution'!
finish: memento withScheduler: aJobScheduler
	"The job is finished, so do any final actions necessary.
	 This method is called within a transaction."

	[self atEnd: memento] assert.
	"do nothing by default."!
!


!Job methodsFor: 'initialize-release'!
initialize

	tagInteger := 0.
	quorumFractionNumerator := 1.
	quorumFractionDenominator := 1.!
!


!Job methodsFor: 'accessing'!
jobId

	^jobId!
jobId: anInteger

	jobId := anInteger.!
originatorIndex

	^originatorIndex!
originatorIndex: anInteger

	originatorIndex := anInteger.!
quorumFraction

	^quorumFractionNumerator / quorumFractionDenominator!
quorumFraction: aFraction
	"This fraction represents the portion of a quorum that this job represents.  When the
	 jobs within a contentionSpace that have the same tag have fractions that add up to 1, they
	 are all allowed to run.  A job with a quorum fraction of 1 has no need to synchronize."

	quorumFractionNumerator := aFraction numerator.
	quorumFractionDenominator := aFraction denominator.!
!
```

```
!Job methodsFor: 'execution'!
restart
	"The job is restarting after a crash.  This method is called within a transaction.  Answer
	a memento that will be passed to #atEnd:, #step:withScheduler:, and #finish: at a later time."

	^nil  "the memento"!
start
	"The job is starting (for the first time, at least as far as what was committed).
	This method is called within a transaction.  Answer a memento that will be
	passed to #atEnd:, #step:withScheduler:, and #finish: at a later time."

	^nil  "the memento"!
step: memento withScheduler: scheduler
	"Step the job.  If an output row is needed, just ask the scheduler for its currentOutputRow.
	This method is called within a transaction.  Answer a new memento to be used in
	successive calls to #atEnd:, #step:withScheduler:, or #finish:.  After a crash, the job will
	be sent #restart to acquire a new memento."

	self subclassResponsibility.
	^memento!
!


!Job methodsFor: 'accessing'!
tag
	"Answer a (lightweight) JobSynchronizationTag that is used to identify jobs that
	must be synchronized together.  Answer nil if no synchronization is required."

	tagInteger == 0 ifTrue: [^nil].
	^JobSynchronizationTag new fromInteger: tagInteger!
tag: aJobSynchronizationTagOrNil
	"Set my (lightweight) tag.  This is used to indicate jobs that must be synchronized together."

	aJobSynchronizationTagOrNil isNil
		ifTrue: [tagInteger := 0]
		ifFalse: [tagInteger := aJobSynchronizationTagOrNil tagInteger]!
tagInteger: anInteger

	tagInteger := anInteger.!
!

-----------------------------------------------------------------------------
Smalltalk defineClass: #OneStepJob
	superclass: #{Job}
	indexedType: #none
	private: false
	instanceVariableNames: ''
	classInstanceVariableNames: ''
	imports: ''
	category: ''!


!OneStepJob methodsFor: 'private: execution'!
atEnd: aMemento
	"Answer whether the job has finished stepping.  If so, the #finish: message will be sent to it.
	This method is called within a transaction.  Normally, this memento was the result of the previous
	execution step, but after a crash, the #restart message is sent to acquire a memento."

	^aMemento!
!


!OneStepJob methodsFor: 'execution'!
executeWith: aContentionSpaceProcess

	self subclassResponsibility.!
!
```

```
!OneStepJob methodsFor: 'private: execution'!
restart
	"The job is restarting after a crash.  This method is called within a transaction.  Answer
	 a memento that will be passed to #atEnd:, #step:withScheduler:, and #finish: at a later time."

	^false!
start
	"The job is starting (for the first time, at least as far as what was committed).
	 This method is called within a transaction.  Answer a memento that will be
	 passed to #atEnd:, #step:withScheduler:, and #finish: at a later time."

	^false!
step: memento withScheduler: scheduler
	"Step the job.  If an output row is needed, just ask the scheduler for its currentOutputRow.
	 This method is called within a transaction.  Answer a new memento to be used in
	 successive calls to #atEnd:, #step:withScheduler:, or #finish:.  After a crash, the job will
	 be sent #restart to acquire a new memento."

	self executeWith: scheduler.
	^true!
!


Smalltalk defineClass: #BouncingJob
	superclass: #{OneStepJob}
	indexedType: #none
	private: false
	instanceVariableNames: '
		bounces <uint32>
		firstBounceTime <uint32>
		lastBounceTime <uint32>
		sumOfSquares <uint32> '
	classInstanceVariableNames: ''
	imports: ''
	category: ''!


!BouncingJob class methodsFor: 'filling & cleaning up'!
collectingStatistics

	CollectingStatistics isNil ifTrue: [
		CollectingStatistics := false].
	^CollectingStatistics!
collectingStatistics: aBoolean
	"When true, this process aggregates the information from all the BouncingJobs that arrive here.
	 When false, the process simply bounces a copy of the job back into the scheduler with updated
	 statistics."

	"BouncingJob collectingStatistics: true"
	"BouncingJob collectingStatistics: false"

	CollectingStatistics := aBoolean.
	CollectingStatistics ifFalse: [
		TotalBounces := 0.
		TotalBounceTimes := 0.
		TotalSquaredBounceTimes := 0.
		EarliestBounceTime := 1e20.
		LatestBounceTime := -1e20.
	].!
```

```
describeStatistics
	"Describe the statistics previously collected."
	"BouncingJob describeStatistics"

	| out columns mean |
	DatabaseSession currentSession transaction: [
		columns := MioSystem currentPersistent maxContentionIndex].
	out := WriteStream on: String new.
	TotalBounces > 1 ifTrue: [
		| popVariance stdDev elapsed |
		mean := (TotalBounceTimes / TotalBounces) asFloat.
		popVariance := (TotalSquaredBounceTimes / TotalBounces) - mean squared.
		stdDev := (popVariance * TotalBounces / (TotalBounces - 1)) sqrt.
		elapsed := LatestBounceTime - EarliestBounceTime max: 1.
		out
			nextPutAll: ', ';
			print: TotalBounces; nextPutAll: ', ';
			print: elapsed; nextPutAll: ', ';
			print: mean; nextPutAll: ', ';
			print: stdDev; nextPutAll: ', ';
			print: (elapsed / TotalBounces) * 1000.0; nextPutAll: ', ';
			print: TotalBounces / elapsed asFloat; cr;
			nextPutAll: 'Proc, jobs exec, time, lat, lat dev, CPU ms/job, jobs/(CPU s)'.
	].
	^out contents!
!


!BouncingJob class methodsFor: 'private'!
initialize

	ObjectMemory removeDependent: self.
	ObjectMemory addDependent: self.!
obsolete

	super obsolete.
	ObjectMemory removeDependent: self!
!


!BouncingJob class methodsFor: 'private: generated'!
ooCodeGenVersion

	^ 1!
ooTypedInstanceVariablesString

	^ '
		bounces <uint32>
		firstBounceTime <uint32>
		lastBounceTime <uint32>
		sumOfSquares <uint32> '!
!


!BouncingJob class methodsFor: 'private'!
random

	Rnd isNil ifTrue: [Rnd := Random new].
	^Rnd!
!


!BouncingJob class methodsFor: 'filling & cleaning up'!
resetStatistics
	"Reset my statistics."
	"BouncingJob resetStatistics"

	TotalBounces := 0.
	TotalBounceTimes := 0.
	TotalSquaredBounceTimes := 0.
	EarliestBounceTime := 1e20.
	LatestBounceTime := -1e20.!
!
```

```
!BouncingJob class methodsFor: 'private'!
update: anAspectSymbol with: aParameter from: aSender

    aSender == ObjectMemory ifTrue: [
        anAspectSymbol == #returnFromSnapshot ifTrue: [
            Rnd := nil]].
    ^super update: anAspectSymbol with: aParameter from: aSender!
!


!BouncingJob class methodsFor: 'EM-Internal'!
_PRAGMA_

    "(defineStatic: #Rnd private: false constant: false category: 'As yet unclassified' initializer: nil)"
    "(defineStatic: #CollectingStatistics private: false constant: false category: 'As yet unclassified' initializer: nil)"
    "(defineStatic: #TotalBounces private: false constant: false category: 'As yet unclassified' initializer: nil)"
    "(defineStatic: #TotalBounceTimes private: false constant: false category: 'As yet unclassified' initializer: nil)"
    "(defineStatic: #TotalSquaredBounceTimes private: false constant: false category: 'As yet unclassified' initializer: nil)"
    "(defineStatic: #EarliestBounceTime private: false constant: false category: 'As yet unclassified' initializer: nil)"
    "(defineStatic: #LatestBounceTime private: false constant: false category: 'As yet unclassified' initializer: nil)"!
!


!BouncingJob methodsFor: 'accessing'!
bounces: int1 firstBounceTime: int2 lastBounceTime: int3 sumOfSquares: int4

    bounces := int1.
    firstBounceTime := int2.
    lastBounceTime := int3.
    sumOfSquares := int4.!
!


!BouncingJob methodsFor: 'execution'!
executeWith: aContentionSpaceProcess
    "I simply add another BouncingJob into a random contention space."

    | now |
    now := Timestamp now asSeconds.  "Don't run tests past midnight."
    bounces = 0
        ifTrue: [
            firstBounceTime := now.
            EarliestBounceTime := EarliestBounceTime min: firstBounceTime]
        ifFalse: [
            sumOfSquares := sumOfSquares + (now - lastBounceTime) squared].
    lastBounceTime := now.
    bounces = 10
        ifTrue: [
            "Don't count the length of time this job sat in the scheduler just prior to statistics collection."
            TotalBounces := TotalBounces + bounces.  "count *complete* trips through contention space processes."
            TotalBounceTimes := TotalBounceTimes + lastBounceTime - firstBounceTime.
            TotalSquaredBounceTimes := TotalSquaredBounceTimes + sumOfSquares.
            LatestBounceTime := LatestBounceTime max: lastBounceTime]
        ifFalse: [
            "We're not compiling statistics yet, so shuffle this job around."
            | targetSpace |
            bounces := bounces + 1.
            targetSpace := (self class random next * aContentionSpaceProcess numberOfContentionSpaces) truncated + 1.
            aContentionSpaceProcess addJob: (self copy
                contentionIndex: targetSpace)].!
!


!BouncingJob methodsFor: 'instance initialization'!
initialize

    super initialize.
    bounces := 0.
    firstBounceTime := 0.
    lastBounceTime := 0.
    sumOfSquares := 0.!
!
```

```
Smalltalk defineClass: #QueueControlJob
	superclass: #{OneStepJob}
	indexedType: #none
	private: false
	instanceVariableNames: '
		command <ooVString> '
	classInstanceVariableNames: ''
	imports: ''
	category: ''!


!QueueControlJob class methodsFor: 'private: generated'!
ooCodeGenVersion

	^ 1!
ooTypedInstanceVariablesString

	^ '
		command <ooVString> '!
!


!QueueControlJob methodsFor: 'accessing'!
command: aString

	command := aString.!
!


!QueueControlJob methodsFor: 'execution'!
executeWith: aContentionSpaceProcess

	Transcript cr; cr; show: '********** Executing command: '''; show: command; nextPutAll: '''.'; cr.
	Compiler evaluate: command.!
!


Smalltalk defineClass: #ReplyToChangeNodeJob
	superclass: #{OneStepJob}
	indexedType: #none
	private: false
	instanceVariableNames: '
		changeNode <ChangeNode>
		newObject <Object> '
	classInstanceVariableNames: ''
	imports: ''
	category: ''!


!ReplyToChangeNodeJob class methodsFor: 'private: generated'!
ooCodeGenVersion

	^ 1!
ooTypedInstanceVariablesString

	^ '
		changeNode <ChangeNode>
		newObject <Object> '!
!


!ReplyToChangeNodeJob methodsFor: 'accessing'!
changeNode: aChangeNode

	changeNode := aChangeNode.!
!
```

```
!ReplyToChangeNodeJob methodsFor: 'execution'!
executeWith: aContentionSpaceProcess

    [changeNode isBusy] assert.

    changeNode isDestinedForDestruction ifTrue: [
        "This ChangeNode was simply disconnecting its object as its final act.  It should now
         be destroyed.  It was already removed from its CompareDictionary."
        changeNode newVersion notNil ifTrue: [
            changeNode newVersion ooDelete.
            changeNode newVersion: nil].
        changeNode oldVersion notNil ifTrue: [
            changeNode oldVersion ooDelete.
            changeNode oldVersion: nil].
        newObject isNil
            ifTrue: [
                "The object has been deleted.  Delete this change node."
                changeNode ooDelete]
            ifFalse: [
                "The object was just finished creating or updating itself.  That must have been
                 happening when the change node was marked for deletion.  Therefore, send
                 another update request to the object (telling it to delete itself)."
                [changeNode isChangedWhileBusy] assert.
                changeNode isChangedWhileBusy: false.
                changeNode object: nil.
                aContentionSpaceProcess addJob: (UpdateObjectJob new
                    contentionIndex: changeNode objectContentionIndex;
                    objectOrNil: newObject;
                    coclusterObject: newObject;
                    record: nil;
                    changeNode: changeNode;
                    changeNodeContentionIndex: contentionIndex)].
        ^self].

    changeNode object: newObject.

    changeNode isChangedWhileBusy
        ifTrue: [
            | dataSource policy |
            dataSource := changeNode dataSource.
            policy := (MioSystem currentPersistent: aContentionSpaceProcess session)
                clusteringPolicyAt: dataSource mappableRoot smalltalkClassName.
            changeNode
                createCatchUpJobIn: aContentionSpaceProcess
                policy: policy]
        ifFalse: [
            changeNode isBusy: false].!
!


!ReplyToChangeNodeJob methodsFor: 'accessing'!
newObject: anObject

    newObject := anObject.!
!


Smalltalk defineClass: #RequestLoadJob
    superclass: #{OneStepJob}
    indexedType: #none
    private: false
    instanceVariableNames: '
        dataSource <DataSource>
        filename <ooVString>
        requestTime <uint64> '
    classInstanceVariableNames: ''
    imports: ''
    category: ''!
```

```
!RequestLoadJob class methodsFor: 'private: generated'!
ooCodeGenVersion

    ^ 1!
ooTypedInstanceVariablesString

    ^ '
        dataSource <DataSource>
        filename <ooVString>
        requestTime <uint64> '!
!


!RequestLoadJob methodsFor: 'accessing'!
dataSource: aDataSource

    dataSource := aDataSource.!
filename: aString

    filename := aString.!
requestTime: milliseconds

    requestTime := milliseconds.!
!

Smalltalk defineClass: #UpdateObjectJob
    superclass: #{OneStepJob}
    indexedType: #none
    private: false
    instanceVariableNames: '
        objectOrNil <Object>
        coclusterObject <Object>
        changeNode <ChangeNode>
        record <ooShortRef(SourceRecord)>
        changeNodeContentionIndex <uint16> '
    classInstanceVariableNames: ''
    imports: ''
    category: ''!


!UpdateObjectJob class methodsFor: 'private: generated'!
ooCodeGenVersion

    ^ 1!
ooTypedInstanceVariablesString

    ^ '
        objectOrNil <Object>
        coclusterObject <Object>
        changeNode <ChangeNode>
        record <ooShortRef(SourceRecord)>
        changeNodeContentionIndex <uint16> '!
!


!UpdateObjectJob methodsFor: 'accessing'!
changeNode: aChangeNode

    changeNode := aChangeNode.!
changeNodeContentionIndex: anInteger

    changeNodeContentionIndex := anInteger.!
coclusterObject: anObject

    coclusterObject := anObject.!
!
```

```
!UpdateObjectJob methodsFor: 'execution'!
executeWith: aContentionSpaceProcess
	"A change node has gotten a new record.  Since we locked the the change node by setting the isBusy flag
	 at the moment the new record was plugged in (into the oldVersion field), we know that nobody else has
	 messed with that field in the interim (if someone saw the isBusy flag during a subsequent load, they would
	 store the new record in the newVersion field and set the isChangedWhileBusy flag)."

	| newObject |
"	[record == changeNode oldVersion] assert."
	record isNil ifTrue: [
		"We need to do a deletion."
		| remoteEntries |
		[objectOrNil notNil] assert: 'A change node should not be marked for deletion until it"s stable'.
"		[changeNode object isNil] assert: 'The object should have been disconnected from the change node already'."
		remoteEntries := objectOrNil remoteIndexEntries.
		objectOrNil changeNode: nil.  "disconnect it immediately."
		objectOrNil isStable
			ifTrue: [
				remoteEntries isEmpty
					ifTrue: [
						"Delete the object now and reply to the change node."
						objectOrNil delete]
					ifFalse: [
						"Start deleting the object's remote index entries."
						| tag |
						tag := aContentionSpaceProcess nextUniqueInteger.
						remoteEntries do: [:entry |
							entry ooClass cacheIndexIfNilForSession: aContentionSpaceProcess session.
							aContentionSpaceProcess addJob: (IndexEntryDeleteJob new
								contentionIndex: entry contentionIndex;
								indexedObject: objectOrNil;
								indexEntry: entry;
								replyTag: tag;
								replyQuorumDenominator: remoteEntries size;
								objectContentionIndex: contentionIndex)].
						objectOrNil beDeleting.
						objectOrNil remoteIndexEntries: Array new]]
			ifFalse: [
				[objectOrNil isDeleting not] assert: 'How could a second delete be requested?'.
				objectOrNil requestDeletion].
		aContentionSpaceProcess addJob: (ReplyToChangeNodeJob new
			contentionIndex: changeNodeContentionIndex;
			changeNode: changeNode;
			newObject: nil).
		^self].

	(objectOrNil isNil or: [objectOrNil isStable])
		ifTrue: [
			| oldTransients plan |
			objectOrNil isNil
				ifTrue: [
					oldTransients := Array new]
				ifFalse: [
					oldTransients := objectOrNil transientIndexEntries.
					[oldTransients size = objectOrNil remoteIndexEntries size] assert].
			plan := record file fileFormat objectConstructionPlanForPolicy: ObjectConstructionPolicy new.
			newObject := plan executeWithRecord: record oldObject: objectOrNil.
			newObject changeNode: changeNode.
			newObject clusterWith: coclusterObject.
			[objectOrNil isNil or: [newObject == objectOrNil]] assert: 'Root object identities must be preserved (currently).'.

			IndexEntryTracker new
				contentionSpaceProcess: aContentionSpaceProcess
				object: newObject
				objectContentionIndex: contentionIndex
				oldTransientsIfKnown: oldTransients]
		ifFalse: [
			newObject := objectOrNil.
			[newObject isDeleting not] assert: 'Why on earth is an object doomed to deletion being updated?'.
			[newObject isReindexing] assert.
			newObject requestReindexing].

	"Send back a job that will
		(A) ensure the changeNode points to the object (not always necessary), and
```

```
            (B) check for pending updates/deletes on the changeNode, queueing jobs as necessary."
        aContentionSpaceProcess addJob: (ReplyToChangeNodeJob new
            contentionIndex: changeNodeContentionIndex;
            changeNode: changeNode;
            newObject: newObject).!
!


!UpdateObjectJob methodsFor: 'accessing'!
objectOrNil: anObject

    objectOrNil := anObject.!
!


!UpdateObjectJob methodsFor: 'deleting'!
ooDelete

    record ooDelete.
    super ooDelete.!
!


!UpdateObjectJob methodsFor: 'copying'!
postCopy

    super postCopy.
    record := record copy.!
!


!UpdateObjectJob methodsFor: 'accessing'!
record: aSourceRecordOrNil

    record := aSourceRecordOrNil.!
!

"-------------------------------------------------------------------------"
Smalltalk defineClass: #RecordLoadingJob
    superclass: #{Job}
    indexedType: #none
    private: false
    instanceVariableNames: '
        filename <ooVString>
        dataSource <DataSource> '
    classInstanceVariableNames: ''
    imports: ''
    category: ''!


!RecordLoadingJob class methodsFor: 'private: generated'!
ooCodeGenVersion

    ^'1'!
ooTypedInstanceVariablesString

    ^'
        filename <ooVString>
        dataSource <DataSource> '!
!


!RecordLoadingJob methodsFor: 'execution'!
atEnd: sourceFileMemento
    "Answer whether the job has finished stepping.  If so, the #finish: message will be sent to it.
     This method is called within a transaction.  Normally, this memento was the result of the previous
     execution step, but after a crash, the #restart message is sent to acquire a memento."

    ^sourceFileMemento atEnd!
!
```

```
!RecordLoadingJob methodsFor: 'accessing'!
dataSource: aDataSource

	dataSource := aDataSource.!
filename: aString

	filename := aString.!
!


!RecordLoadingJob methodsFor: 'execution'!
finish: sourceFileMemento withScheduler: aJobScheduler
	"The job is finished, so do any final actions necessary.
	 This method is called within a transaction."

	[self atEnd: sourceFileMemento] assert.
	self snapshot finish: sourceFileMemento withScheduler: aJobScheduler.!
restart
	"The job is restarting after a crash.  This method is called within a transaction.  Answer
	 a memento that will be passed to #atEnd:, #step:withScheduler:, and #finish: at a later time."

	^self snapshot sourceFile!
!


!RecordLoadingJob methodsFor: 'accessing'!
snapshot

	^dataSource snapshots last!
!


!RecordLoadingJob methodsFor: 'execution'!
start
	"The job is starting (for the first time, at least as far as what was committed).
	 This method is called within a transaction.  Answer a memento that will be
	 passed to #atEnd:, #step:withScheduler:, and #finish: at a later time."

	| session sys tempMap formatMap formatCopy newSnapshot |
	session := dataSource ooContainer session.
	sys := MioSystem currentPersistent: session.

	tempMap := IdentityDictionary new.
	sys schema deepCopyCreatingHomomorphism: tempMap.
	formatMap := IdentityDictionary new.
	tempMap keysDo: [:stub |
		formatMap at: stub put: stub].
	formatMap at: dataSource put: dataSource.
	formatCopy := dataSource currentFormat deepCopyCreatingHomomorphism: formatMap.

	newSnapshot := DataSnapshot
		dataSource: dataSource
		fileName: filename
		format: formatCopy.
	dataSource cluster: newSnapshot.
	dataSource addSnapshot: newSnapshot.
	newSnapshot createRecordsContainerInColumn: contentionIndex.

	^self snapshot sourceFile!
step: sourceFileMemento withScheduler: scheduler
	"Step the job.  If an output row is needed, just ask the scheduler for its currentOutputRow.
	 This method is called within a transaction.  Answer a new memento to be used in
	 successive calls to #atEnd:, #step:withScheduler:, or #finish:.  After a crash, the job will
	 be sent #restart to acquire a new memento."

	self snapshot loadNextRecordFrom: sourceFileMemento jobScheduler: scheduler.
	^sourceFileMemento!
!
```

```
Smalltalk defineClass: #JobBuffer
	superclass: #{Core.Object}
	indexedType: #none
	private: false
	instanceVariableNames: '
		firstJobId <uint32>
		nextJobId <uint32>
		firstJobIndex <uint32>
		numberedJobs <ooVArray(ooShortRef(Job))>
		toWrite <ooTransient> '
	classInstanceVariableNames: ''
	imports: ''
	category: ''!


!JobBuffer class methodsFor: 'instance creation'!
new

	^super new initialize!
!


!JobBuffer class methodsFor: 'private: generated'!
ooCodeGenVersion

	^ 1!
ooTypedInstanceVariablesString

	^'
		firstJobId <uint32>
		nextJobId <uint32>
		firstJobIndex <uint32>
		numberedJobs <ooVArray(ooShortRef(Job))>
		toWrite <ooTransient> '!
!


!JobBuffer methodsFor: 'accessing'!
addJob: aJob

	aJob ooIsPersistent ifTrue: [
		self error: 'Jobs to be transmitted must not be persistent'].
	toWrite isNil ifTrue: [toWrite := OrderedCollection new].
	toWrite addLast: aJob.!
!


!JobBuffer methodsFor: 'ui support'!
currentJobCount
	"Used for saturation testing."

	^nextJobId  "Assumes 32-bit wrapping won't happen during test.".!
!
```

```
!JobBuffer methodsFor: 'accessing'!
deleteJobsUpTo: jobId
	"Delete all jobs up to and including the one with the given jobId.  This message might
	 arrive more than once, so be careful to ignore it if it specifies a job that has already
	 been deleted.  It should never be more than two billion jobs out of date, otherwise the
	 update stream processor may fail.  Answer the actual number of jobs deleted."

	| firstToKeep howManyToKill |
	firstToKeep := jobId + 1 "bitAnd: 16rFFFFFFFF".
	howManyToKill := firstToKeep - firstJobId bitAnd: 16rFFFFFFFF.
	howManyToKill > 16r80000000 ifTrue: [
		"A request to delete more than two billion jobs is (probably) really a request to
		 delete a negative number of jobs, i.e., these jobs have already been deleted."
		^0].
	numberedJobs accessIn: [
		| size |
		size := numberedJobs size.
		howManyToKill timesRepeat: [
			| job |
			job := numberedJobs at: firstJobIndex.
			numberedJobs at: firstJobIndex put: nil.
			job ooDelete.
			firstJobIndex := firstJobIndex \\ size + 1].
		firstJobId := firstJobId + howManyToKill bitAnd: 16rFFFFFFFF].
	self ooUpdate.  "Potential Objectivity bug"
	^howManyToKill!
!


!JobBuffer methodsFor: 'initialize-release'!
initialize

	firstJobId := 1.
	nextJobId := 1.
	firstJobIndex := 1.
	numberedJobs := OoVArray new: 100.!
!


!JobBuffer methodsFor: 'accessing'!
prepareToCommit
	"Write persistent copies of the new jobs into the database.  Answer the (original) transient jobs for
	 which copies were written.  These will be transmitted (after the commit) to the host responsible for
	 executing them."

	| size result |
	(toWrite isNil or: [toWrite isEmpty]) ifTrue: [^#()].
	size := toWrite size.
	result := self prepareToCommitHelper.
	[size = result size] assert.
	^result!
!
```

```
!JobBuffer methodsFor: 'private-helper'!
prepareToCommitHelper
	"Write persistent copies of the new jobs into the database.  Answer the (original) transient jobs for
	 which copies were written.  These will be transmitted (after the commit) to the host responsible for
	 executing them."

	| size capacity position result |
	[toWrite notNil and: [toWrite notEmpty]] assert.

	numberedJobs accessIn: [
		size := nextJobId - firstJobId bitAnd: 16rFFFFFFFF.
		capacity := numberedJobs size.
		size + toWrite size >= capacity ifTrue: [
			"Not enough room to add all the jobs.  Enlarge the buffer to the new required capacity * 1.5."
			| newCapacity growth lastIndex |
			newCapacity := (size + toWrite size) * 3 // 2 + 2.
			growth := newCapacity - capacity.
			numberedJobs changeSizeTo: newCapacity.
			self ooUpdate.  "Objectivity bug"
			lastIndex := firstJobIndex + size - 1.
			lastIndex > capacity ifTrue: [
				"The data wrapped around past the end of the old buffer.  Move that part of it to
				 the end of the new buffer."
				numberedJobs
					replaceFrom: firstJobIndex + growth
					to: newCapacity
					with: numberedJobs
					startingAt: firstJobIndex.
				numberedJobs
					atAll: (firstJobIndex to: firstJobIndex + growth - 1)
					put: nil.
				firstJobIndex := firstJobIndex + growth].
			capacity := newCapacity].
		size + toWrite size >= capacity ifTrue: [self error: 'Bug in circular buffer growth algorithm'].
		position := firstJobIndex + nextJobId - firstJobId.
		position := (position - 1) \\ capacity + 1.
		toWrite do: [:job |
			job jobId: nextJobId.
			nextJobId := nextJobId + 1 bitAnd: 16rFFFFFFFF.
			[(numberedJobs at: position) isNil] assert.
			numberedJobs at: position put: job copy.
			position := position \\ capacity + 1]].

	result := toWrite.
	toWrite := nil.
	^result!
!


!JobBuffer methodsFor: 'ui support'!
resetAllJobs
	"This is extremely dangerous, and is only to be used for testing.  Delete all jobs and
	 reset my job numbering counters."

	numberedJobs do: [:j |
		j == nil ifFalse: [j ooDelete]].
	numberedJobs atAllPut: nil.
	firstJobId := 1.
	nextJobId := 1.
	firstJobIndex := 1.
	numberedJobs := OoVArray new: 100.
	toWrite := nil.!
!
```

```
!JobBuffer methodsFor: 'recovery'!
retransmitAllViaChannel: jobChannel
	"This is part of the recovery algorithm.  Retransmit all jobs still in this buffer.  A contention space
	 will simply ignore jobs that have already been executed.  This must be called within an MROW
	 transaction."

	| count usp lastExecuted |
	[jobChannel wasRecovered] assert.

	"First, check with the contention space at the other end (via Objectivity) to see how far it got before the crash."
	usp := UpdateStreamProcessor named: 'Main' inSession: self ooContainer session.
	lastExecuted := (usp participantAt: jobChannel remoteId) lastJobsExecuted at: jobChannel myId.

	"Don't retransmit any of the completed jobs."
	self deleteJobsUpTo: lastExecuted.

	"Transmit everything after the point at which it stopped executing."
	count := nextJobId - firstJobId bitAnd: 16rFFFFFFFF.
	USPControllerService current at: 1 log: 'Retransmitting ', count printString, ' jobs from ', jobChannel myId printString, ' to ', jobChannel remoteId printString.
	jobChannel wasRecovered: false.
	jobChannel start.
	numberedJobs accessIn: [
		| index clump |
		index := firstJobIndex.
		clump := OrderedCollection new: 500.
		count timesRepeat: [
			| job |
			job := numberedJobs at: index.
			[job notNil] assert.
			clump addLast: job copy.
			clump size = 1000 ifTrue: [
				jobChannel nextPutAll: clump.
				clump removeLast: clump size].
			index := index + 1.
			index > numberedJobs size ifTrue: [index := 1]].
		jobChannel nextPutAll: clump.
	].!
!

Smalltalk defineClass: #SaturationTestJobBuffer
	superclass: #{JobBuffer}
	indexedType: #none
	private: false
	instanceVariableNames: ''
	classInstanceVariableNames: ''
	imports: ''
	category: ''!

!SaturationTestJobBuffer methodsFor: 'accessing'!
addJob: aJob
	"Note - we don't actually store the job for the saturation test."

	aJob ooIsPersistent ifTrue: [
		self error: 'Jobs to be transmitted must not be persistent'].
	toWrite isNil ifTrue: [toWrite := OrderedCollection new].
	toWrite addLast: aJob.!
deleteJobsUpTo: jobId
	"*** Saturation test requires that I not actually persist my jobs."

	| firstToKeep howManyToKill |
	firstToKeep := jobId + 1 "bitAnd: 16rFFFFFFFF".
	howManyToKill := firstToKeep - firstJobId bitAnd: 16rFFFFFFFF.
	howManyToKill > 16r80000000 ifTrue: [
		"A request to delete more than two billion jobs is (probably) really a request to
		 delete a negative number of jobs, i.e., these jobs have already been deleted."
		^0].
	firstJobIndex := 1.
	firstJobId := firstJobId + howManyToKill bitAnd: 16rFFFFFFFF.
	^howManyToKill!
!
```

```
!SaturationTestJobBuffer methodsFor: 'recovery'!
initialize

	super initialize.
	numberedJobs := OoVArray new.  "This instvar should be unused."!
!


!SaturationTestJobBuffer methodsFor: 'private-helper'!
prepareToCommitHelper
	"Write persistent copies of the new jobs into the database.  Answer the (original) transient jobs for
	 which copies were written.  These will be transmitted (after the commit) to the host responsible for
	 executing them."

	"Don't write the jobs to the database.  This is a saturation test."

	| result |
	[toWrite notNil and: [toWrite notEmpty]] assert.

	toWrite do: [:job |
		job jobId: nextJobId.
		nextJobId := nextJobId + 1 bitAnd: 16rFFFFFFFF].

	result := toWrite.
	toWrite := nil.
	^result!
!


!SaturationTestJobBuffer methodsFor: 'recovery'!
retransmitAllViaChannel: jobChannel
	"This is part of the recovery algorithm.  Retransmit all jobs still in this buffer.  A contention space
	 will simply ignore jobs that have already been executed.  This must be called within a transaction."
	"The saturation test supresses persistent storage of jobs with the producer.  Do nothing."

	| usp lastExecuted |
	[jobChannel wasRecovered] assert.
	"First, check with the contention space at the other end (via Objectivity) to see how far it got before the crash."
	usp := UpdateStreamProcessor named: 'Main' inSession: self ooContainer session.
	lastExecuted := (usp participantAt: jobChannel remoteId) lastJobsExecuted at: jobChannel myId.

	"Reset my numbering to correspond with the last actually executed job."
	self deleteJobsUpTo: lastExecuted.
	nextJobId := firstJobId.  "don't worry about reusing the id's - the jobs were lost forever."

	USPControllerService current at: 1 log: '(Not retransmitting anything, due to saturation test)'.
	jobChannel wasRecovered: false.
	jobChannel start.!
!


Smalltalk defineClass: #JobChannel
	superclass: #{Core.Object}
	indexedType: #none
	private: false
	instanceVariableNames: 'outQueue inQueue socket writerProcess readerProcess myId remoteId isConnecting activator passivator onConnect onDisconnect
wasRecovered '
	classInstanceVariableNames: ''
	imports: ''
	category: ''!


!JobChannel class methodsFor: 'instance creation'!
new

	^super new initialize!
!
```

```
!JobChannel class methodsFor: 'TEST'!
test1
	"A simple test with two JobChannels connected together via a pair of sockets."
	"JobChannel test1"

	| pair s1 s2 ch1 ch2 result |
	pair := SocketAccessor openPair.
	s1 := pair first.
	s2 := pair last.
	ch1 := self new socket: s1; myId: 1.
	ch2 := self new socket: s2; myId: 2.
	ch1 start.
	ch2 start.
	ch1 nextPut: #(1 2 3); nextPut: #(#Test).
	ch2 nextPut: #('Hello' #[10]).
	(Delay forMilliseconds: 1000) wait.

	[ch1 isEmpty not] assert.
	[ch2 isEmpty not] assert.
	result := ch2 next.
	[result = #(1 2 3)] assert.
	[ch2 isEmpty not] assert.
	result := ch2 next.
	[result = #(#Test)] assert.
	[ch2 isEmpty] assert.

	result := ch1 next.
	[result = #('Hello' #[10])] assert.
	[ch1 isEmpty] assert.
	ch1 stop.
	ch2 stop.!
!
```

```
!JobChannel methodsFor: 'private looping'!
doReading
    "Loop forever, reading from the socket.  Stop when my socket closes (or is replaced by nil)."

    | sock read sizeHolder sizeBytes size |
    (sock := socket) isNil ifTrue: [^self].
    read := sock readStream binary.
    sizeHolder := UninterpretedBytes new: 4.
    sizeBytes := ByteArray new: 4.
    [
        remoteId := read nextLong.
        [remoteId isInteger] assert.
        onConnect notNil ifTrue: [onConnect value: self].
        [
            | bytes |
            read next: 4 into: sizeBytes startingAt: 1.
            sizeHolder replaceBytesFrom: 1 to: 4 with: sizeBytes startingAt: 1.
            socket isNil ifTrue: [^self].
            size := sizeHolder unsignedLongAt: 1.
            bytes := read next: size.
            socket isNil ifTrue: [^self].
            inQueue nextPut: bytes.
        ] repeat.
    ]
        on:
            Stream endOfStreamSignal,
            Stream incompleteNextCountSignal,
            OSErrorHolder peerFaultSignal,
            OSErrorHolder unpreparedOperationSignal,
            OSErrorHolder unsupportedOperationSignal
        do: [:ex |
            | p |
            (p := writerProcess) notNil ifTrue: [p terminate].
            onDisconnect notNil ifTrue: [onDisconnect value: self].
            ^self].!
doWriting
    "Loop forever, writing to the socket.  Stop when my socket closes (or is replaced by nil)."

    | sock write sizeHolder |
    (sock := socket) isNil ifTrue: [^self].
    write := sock writeStream binary.
    sizeHolder := UninterpretedBytes new: 4.
    [
        write nextLongPut: myId; commit.
        [
            | object bytes |
            object := outQueue next.
            bytes := passivator convert: object.
            socket isNil ifTrue: [^self].
            sizeHolder unsignedLongAt: 1 put: bytes size.
            write nextPutAll: sizeHolder asByteArray.
            write nextPutAll: bytes.
            socket isNil ifTrue: [^self].
            outQueue isEmpty ifTrue: [
                "Transcript cr; show: 'Short packet: ', write position printString."
                write commit].
        ] repeat.
    ]
        on:
            Stream endOfStreamSignal,
            Stream incompleteNextCountSignal,
            OSErrorHolder peerFaultSignal,
            OSErrorHolder unpreparedOperationSignal,
            OSErrorHolder unsupportedOperationSignal
        do: [:ex |
            "The disconnect action is taken when the *reader* process notices the socket is closed."
            ^self].!
!
```

```
!JobChannel methodsFor: 'accessing'!
hasFirmConnection

	isConnecting ifTrue: [^false].
	wasRecovered ifTrue: [^false].
	(socket isNil or: [readerProcess isNil or: [writerProcess isNil]]) ifTrue: [^false].
	[
		socket primGetPeer
	]
		on:
			OSErrorHolder peerFaultSignal,
			OSErrorHolder unpreparedOperationSignal
		do: [:ex |
			self stop.
			^false].

	^true!
!


!JobChannel methodsFor: 'initialize-release'!
initialize

	outQueue := SharedQueue new.
	inQueue := SharedQueue new.
	isConnecting := false.
	wasRecovered := false.!
!


!JobChannel methodsFor: 'accessing'!
isEmpty

	^inQueue isEmpty!
myId

	^myId!
myId: anInteger

	myId := anInteger.!
next
	"This is only valid if there are no references to persistent objects coming through this channel.  If there
	 are persistent object references, you should use #nextUsingSession:."

	^self nextUsingSession: nil!
nextPut: anObject
	"Push the object onto my outgoing objects queue."

	passivator notNil ifTrue: [
		"There's no way to write at this time, so assume the recovery procedure will transmit this job eventually."
		outQueue nextPut: anObject].!
nextPutAll: aCollectionOfObjects
	"Push the objects onto my outgoing objects queue."

	passivator notNil ifTrue: [
		"There's no way to write at this time, so assume the recovery procedure will transmit this job eventually."
		outQueue nextPutAll: aCollectionOfObjects].!
nextUsingSession: aSession
	"Decode the next object from the inQueue.  We postpone decoding these messages until the object
	 is actually requested (i.e., now), so that we can correctly connect to Objectivity/DB objects.  Note
	 that the activator requires an OoSession in order to decode such objects."

	| bytes |
	bytes := inQueue next.
	activator isNil ifTrue: [self error: 'Activator was destroyed with data in channel'].
	^activator
		session: aSession;
		convert: bytes!
onConnect: aBlock
	"The block takes myself as an argument.  It is invoked once the remoteId has been received."

	onConnect := aBlock.!
```

```
onDisconnect: aBlock
	"The block takes myself as an argument.  It is invoked when the connection is broken."

	onDisconnect := aBlock.!
!


!JobChannel methodsFor: 'printing'!
printOn: aStream

	aStream
		nextPutAll: 'JobChannel(sock=';
		print: (socket notNil ifTrue: [socket getName]);
		nextPutAll: ', writer ';
		nextPutAll: (writerProcess isNil ifTrue: ['not '] ifFalse: ['']);
		nextPutAll: 'active, reader ';
		nextPutAll: (readerProcess isNil ifTrue: ['not '] ifFalse: ['']);
		nextPutAll: 'active, inQueue=';
		print: inQueue size;
		nextPutAll: ', outQueue=';
		print: outQueue size;
		nextPutAll: ')'.!
!


!JobChannel methodsFor: 'accessing'!
privateReestablishIfNecessaryFor: anUpdateStreamProcessor
	"If the socket has been closed, attempt to reestablish a connection with the target job execution process.
	 Assume this is called within an MROW transaction."

	| otherSpace otherIP otherPort |

	wasRecovered := false.  "just in case."
	isConnecting := true.
	self stop.
	otherSpace := anUpdateStreamProcessor participantAt: remoteId.
	otherIP := otherSpace hostName.
	otherPort := otherSpace port.
	(otherIP notEmpty and: [otherPort ~= 0]) ifTrue: [
		[ "fork"
			[ "ifCurtailed:"
				[ "on:do:"
					USPControllerService current at: 2 log: 'Reestablishing link from ', myId printString,
						' to ', remoteId printString, '...'.
					socket := SocketAccessor
						newTCPclientToHost: otherIP
						port: otherPort.
					isConnecting := false.
					wasRecovered := true.
					USPControllerService current at: 2 log: '...done reconnecting ', myId printString,
						' to ', remoteId printString, '.'.
				]
					on: OSErrorHolder peerFaultSignal
					do: [:ex |
						USPControllerService current at: 2 log: '...aborting reconnection attempt from ', myId printString,
							' to ', remoteId printString, '.'.
						self stop.
						isConnecting := false.
						ex return: nil].
			] ifCurtailed: [
				isConnecting := false.
				self stop.
			].
		] forkAt: Processor activeProcess priority - 1.
	].!
reestablishIfNecessaryFor: anUpdateStreamProcessor
	"If the socket has been closed, attempt to reestablish a connection with the target job execution process.
	 Assume this is called within an MROW transaction."

	isConnecting ifTrue: [^self].
	wasRecovered ifTrue: [^self].
	(socket isNil or: [readerProcess isNil or: [writerProcess isNil]]) ifFalse: [^self].

	^self privateReestablishIfNecessaryFor: anUpdateStreamProcessor!
```

```
remoteId

	^remoteId!
remoteId: anInteger

	remoteId := anInteger.!
shouldCheck
	"Answer whether this channel should be processed now.  Does not need a transaction."

	isConnecting ifTrue: [^false].
	wasRecovered ifTrue: [^true].
	(socket isNil or: [readerProcess isNil or: [writerProcess isNil]]) ifTrue: [^true].
	inQueue isEmpty ifFalse: [^true].

	^false!
socket: aSocketAccessor

	socket := aSocketAccessor.!
start
	"Start up the processes for dealing with jobs and replies.  Note that if there is no
	 valid socket, neither process will start up.  It is the sender's responsibility to attempt
	 to reestablish a connection to a failed socket (because of the need to access the
	 database to find IP and port numbers)."

	socket isNil ifTrue: [^self].  "Do nothing in this case."
	[writerProcess isNil] assert.
	[readerProcess isNil] assert.
	passivator := MiosoftPassivator new.
	activator := MiosoftActivator new.
	writerProcess := Process
		forBlock: [[self doWriting] ensure: [writerProcess := nil]]
		priority: Processor userSchedulingPriority - 2.
	readerProcess := Process
		forBlock: [[self doReading] ensure: [readerProcess := nil]]
		priority: Processor userSchedulingPriority - 2.
	writerProcess resume.
	readerProcess resume.!
stop
	"Stop the processes that deal with jobs and replies.  Wait until the processes have
	 actually stopped."

	| sock p |
	sock := socket.
	socket := nil.
	(p := writerProcess) notNil ifTrue: [p terminate. writerProcess := nil].
	(p := readerProcess) notNil ifTrue: [p terminate.  readerProcess := nil].
	sock notNil ifTrue: [
		"sock shutdown: 2."  "Close read&write directions."
		sock close].
	[outQueue isEmpty] whileFalse: [outQueue next].
	outQueue := SharedQueue new.!
wasRecovered
	"Answer whether this channel has been connected.  This lets us know when we need to
	 retransmit all jobs through a new socket."

	^wasRecovered!
wasRecovered: aBoolean

	wasRecovered := aBoolean.!
!

"-------------------------------------------------------------------------"
Smalltalk defineClass: #SaturationTestJobChannel
	superclass: #{JobChannel}
	indexedType: #none
	private: false
	instanceVariableNames: 'jobGenerator '
	classInstanceVariableNames: ''
	imports: ''
	category: ''!
```

```
!SaturationTestJobChannel methodsFor: 'accessing'!
jobGenerator

	^jobGenerator!
jobGenerator: aBlock
	"This block says how to generate a thousand jobs (when the number of outstanding
	 jobs drops below the low water mark)."

	jobGenerator := aBlock.!
!

"-------------------------------------------------------------------------"
Smalltalk defineClass: #JobSynchronizationTag
	superclass: #{Core.Object}
	indexedType: #none
	private: false
	instanceVariableNames: 'tagInteger hash '
	classInstanceVariableNames: ''
	imports: ''
	category: ''!


!JobSynchronizationTag methodsFor: 'comparing'!
= another

	another class == self class ifFalse: [^false].
	^self tagInteger = another tagInteger!
!


!JobSynchronizationTag methodsFor: 'accessing'!
fromInteger: int

	tagInteger := int.
	hash := 0.
	1 to: 8 do: [:i |
		hash := (hash + (tagInteger digitAt: i)) timesRandomMultiplier].!
!


!JobSynchronizationTag methodsFor: 'comparing'!
hash

	^hash!
!


!JobSynchronizationTag methodsFor: 'printing'!
printOn: aStream

	aStream nextPutAll: '{tag='; print: tagInteger; nextPutAll: '}'.!
!


!JobSynchronizationTag methodsFor: 'accessing'!
tagInteger

	^tagInteger!
!

"-------------------------------------------------------------------------"
Smalltalk defineClass: #USPControllerService
	superclass: #{UI.Model}
	indexedType: #none
	private: false
	instanceVariableNames: 'socket outputSocket outputLock responder runningHosts hostsThatReplied processes session logProtect logEnable '
	classInstanceVariableNames: ''
	imports: ''
	category: ''!
```

```
!USPControllerService class methodsFor: 'class initialization'!
aboutToQuit

    self current shutdown.!
!


!USPControllerService class methodsFor: 'instance creation'!
current

    Current isNil ifTrue: [self initialize].
    ^Current!
!


!USPControllerService class methodsFor: 'class initialization'!
initialize
    "USPControllerService initialize"

    ObjectMemory removeDependent: self.
    ObjectMemory addDependent: self.
    Current notNil ifTrue: [
        [Current shutdown] on: Object errorSignal do: [:ex | ex return].
        (Delay forMilliseconds: 500) wait].
    Current := super new initialize startUp.
    LogAccess := Semaphore forMutualExclusion.!
!


!USPControllerService class methodsFor: 'class accessing'!
magicPort
    "Answer the port to use for controlling the USP (via datagrams)."

    ^13793!
!


!USPControllerService class methodsFor: 'instance creation'!
new

    self shouldNotImplement.  "Use #current."!
!


!USPControllerService class methodsFor: 'class initialization'!
obsolete
    "This class is being removed from the system.  Destroy my instance and unregister me."

    self == USPControllerService ifTrue: [
        Current notNil ifTrue: [Current shutdown].
        ObjectMemory removeDependent: self].
    super obsolete!
returnFromSnapshot

    MiosoftWindowsSupport runDosAndWait: 'cmd /c oocleanup -local c:\miosoft\bin\currentdb'.
    self current startUp.!
update: anAspectSymbol with: aParameter from: aSender

    aSender == ObjectMemory ifFalse: [^self].
    anAspectSymbol == #aboutToQuit ifTrue: [
        self aboutToQuit].
    anAspectSymbol == #returnFromSnapshot ifTrue: [
        self returnFromSnapshot].!
!


!USPControllerService class methodsFor: 'EM-Internal'!
_PRAGMA_

    "(defineStatic: #Current private: false constant: false category: 'As yet unclassified' initializer: nil)"
    "(defineStatic: #LogAccess private: false constant: false category: 'As yet unclassified' initializer: nil)"!
!
```

```
!USPControllerService methodsFor: 'private - responder'!
addHost: aString

	((runningHosts includes: aString)
		and: [hostsThatReplied includes: aString])
		ifFalse: [
			hostsThatReplied add: aString.
			runningHosts add: aString.
			self changed: #runningHosts].!
!


!USPControllerService methodsFor: 'accessing'!
addProcess: aProcess

	aProcess addDependent: self.
	processes at: aProcess myId put: aProcess.!
!


!USPControllerService methodsFor: 'private - responder'!
at: indent log: aString
	"Log something to the Transcript (when enabled)."

	logEnable ifTrue: [
		Transcript dependents first topComponent notNil ifTrue: [
			logProtect critical: [
				Transcript crtab: indent; show: aString]]].!
!


!USPControllerService methodsFor: 'private - i/o'!
broadcast: aString
	"Send this message to every awake host on the local network.  Not all
	 messages will be received, but that's ok."

	| netAddr |
	[socket notNil] assert.
	self at: 0 log: 'Broadcast: "', aString, '"'.
	"netAddr := IPSocketAddress
		hostAddress: (IPSocketAddress broadcastAddressForNet: socket getName networkAddress)
		port: self class magicPort."
	netAddr := IPSocketAddress
		hostAddress: #[255 255 255 255]  "non-portable Windows crap"
		port: self class magicPort.

	outputLock critical: [
		outputSocket
			setOptionsLevel: SocketAccessor solSocket
			name: 32  " = SocketAccessor soBroadcast"
			value: 1.  "enable broadcasting."
		outputSocket writeWait.
		outputSocket sendTo: netAddr buffer: aString].!
!
```

```
!USPControllerService methodsFor: 'private - responder'!
handleMessage: msgString from: originatorHostName
    "Handle one message."

    self at: 0 log: 'Received: "', msgString, '" from ', originatorHostName.
    ('Running on *' match: msgString) ifTrue: [
        | str host ids |
        str := msgString readStream.
        str skip: 'Running on ' size.
        host := str nextLine.
        self addHost: host.
        ids := Set new.
        [str atEnd] whileFalse: [
            | line id proc |
            line := str nextLine readStream.
            id := Integer readFrom: line.
            ids add: id.
            line next = Character space ifFalse: [self error: 'Syntax error in status update'].
            proc := self processAt: id.
            proc isNil ifTrue: [
                proc := RemoteProcess new
                    initializeForIndex: id
                    host: host
                    status: 'should not see this'.
                self addProcess: proc].
            proc isLocal ifFalse: [
                proc status: line upToEnd]].
        processes values do: [:p |
            (p host = host and: [(ids includes: p myId) not]) ifTrue: [
                self removeProcessId: p myId]].
        self changed: #status.
        ^self].
    ('Reboot image' match: msgString) ifTrue: [
        [
            self shutdown.
            Win32SystemSupport CreateProcess: nil arguments: 'cmd /c start c:\miosoft\miocon'.
            [(Delay forSeconds: 10) wait. ObjectMemory quitPrimitive] fork.
            [ObjectMemory quit] fork.
        ] fork.
        ^self.
    ].
    ('Shutting down *' match: msgString) ifTrue: [
        ^self removeHost: (msgString readStream skip: 'Shutting down ' size; nextLine)].
    ('Resend greeting' match: msgString) ifTrue: [
        ^self send: self statusMessage toHost: originatorHostName].
    ('Install process *' match: msgString) ifTrue: [
        ^[self installLocalProcessId: (msgString readStream skip: 'Install process ' size; nextLine) asNumber] fork].
    ('Start process *' match: msgString) ifTrue: [
        ^self startProcessId: (msgString readStream skip: 'Start process ' size; nextLine) asNumber].
    ('Stop process *' match: msgString) ifTrue: [
        ^self stopProcessId: (msgString readStream skip: 'Stop process ' size; nextLine) asNumber].
    ('Log *' match: msgString) ifTrue: [
        ^self log: (msgString readStream skip: 'Log ' size; nextLine) from: originatorHostName].
    ('Sync to *' match: msgString) ifTrue: [
        ^self syncTime: (msgString readStream skip: 'Sync to ' size; nextLine)].
    ('Execute *' match: msgString) ifTrue: [
        ^[Compiler evaluate: (msgString readStream skip: 'Execute ' size; nextLine)] fork].
    ('Snoop' match: msgString) ifTrue: [
        'screenshot' asFilename writeStream binary
            nextPutAll: (Passivator convert: (Screen default completeContentsOfArea: Screen default bounds));
            close.
        ^self send: 'Snoop done' toHost: originatorHostName].
    ('Snoop done' match: msgString) ifTrue: [
        ^self changed: #snoopDone with: originatorHostName].

    self error: 'Unrecognized message'.!
!


!USPControllerService methodsFor: 'accessing'!
hostsThatReplied

    ^hostsThatReplied!
!
```

```
!USPControllerService methodsFor: 'initialize-release'!
initialize

	hostsThatReplied := Set new.
	runningHosts := Set new.
	processes := Dictionary new.  "{id -> (ParticipantProcess | RemoteProcess)}"

	outputLock := Semaphore forMutualExclusion.
	logProtect := Semaphore forMutualExclusion.
	logEnable := true.!
!


!USPControllerService methodsFor: 'private - responder'!
installLocalProcessId: index
	"Install a consumer or producer process on this host."

	| proc |
	proc := self processAt: index.
	proc notNil ifTrue: [
		^self log: 'Ignored -- that process is already installed' from: ''].

	self session transactionMROW: [
		| usp |
		usp := UpdateStreamProcessor named: 'Main' inSession: self session.
		index <= usp numberOfContentionSpaces
			ifTrue: [proc := ContentionSpaceProcess new]
			ifFalse: [proc := SaturationTestProducerProcess new].  "Hooked for saturation testing."
		proc initializeForIndex: index.
		self addProcess: proc].!
installProcessId: anInteger onHost: hostName
	"Install a consumer or producer process on this machine."

	self send: 'Install process ', anInteger printString toHost: hostName.!
!


!USPControllerService methodsFor: 'accessing'!
log: aString from: originatingHost
	"Something interesting has happenned on the given host.  Report it to interested parties."

	self changed: #log with: originatingHost -> aString.!
!


!USPControllerService methodsFor: 'private - responder'!
logEnable: aBoolean
	"USPControllerService current logEnable: true"

	logEnable := aBoolean.!
!


!USPControllerService methodsFor: 'accessing'!
processAt: id

	^processes at: id ifAbsent: [nil]!
!
```

```
!USPControllerService methodsFor: 'private - responder'!
removeHost: aString

	(runningHosts includes: aString) ifTrue: [
		| anyKilled |
		runningHosts remove: aString.
		hostsThatReplied remove: aString.
		anyKilled := false.
		processes values do: [:p |
			p host = aString ifTrue: [
				p isLocal ifTrue: [p stop].
				self removeProcessId: p myId.
				anyKilled := true]].
		anyKilled ifTrue: [self changed: #status].
		self changed: #runningHosts].!
!


!USPControllerService methodsFor: 'accessing'!
removeProcessId: processId

	(processes includesKey: processId) ifTrue: [
		(processes at: processId) removeDependent: self.
		processes removeKey: processId ifAbsent: []].!
resetHostsThatReplied

	hostsThatReplied := Set new.!
!


!USPControllerService methodsFor: 'private - responder'!
responderLoop
	"Repeat forever, responding to messages from my peers."

	| buffer |
	buffer := ByteArray new: 512.
	[
		[
			| size msg sock neighbour |
			socket readWait.
			(sock := socket) isNil ifTrue: [^self].
			neighbour := IPSocketAddress new.
			size := sock
				receiveFrom: neighbour
				buffer: buffer
				start: 1
				for: buffer size
				flags: 0.
			msg := (buffer copyFrom: 1 to: size) asString.
			[
				self
					handleMessage: msg
					from: neighbour hostName
			] on: Object errorSignal do: [:ex |
				self
					at: 0 log: 'ERROR in USPControllerService:';
					at: 0 log: '    ', ex errorString, "";
					at: 0 log: '    message = ', msg, "".
				InputState default shiftDown ifTrue: [self halt].
				ex return.
			].
		] repeat.
	] ensure: [responder := nil].!
!


!USPControllerService methodsFor: 'accessing'!
runningHosts

	^runningHosts asSortedCollection asArray!
!
```

```
!USPControllerService methodsFor: 'private - i/o'!
send: aString toHost: hostName
	"Send a message to the given host."

	| addr |
	[socket notNil] assert.
	self at: 0 log: 'Send: "', aString, '" to ', hostName.
	addr := IPSocketAddress
		hostName: hostName
		port: self class magicPort.

	outputLock critical: [
		outputSocket
			setOptionsLevel: SocketAccessor solSocket
			name: 32  " = SocketAccessor soBroadcast"
			value: 0.  "disable broadcasting."
		outputSocket writeWait.
		outputSocket sendTo: addr buffer: aString].!
!


!USPControllerService methodsFor: 'accessing'!
session

	(session isNil or: [session isOpen not]) ifTrue: [
		[
			session := OoSession open: 'currentdb'.
			session rpcTimeout: 5000; useIndex: true; recover: false.
		] on: Object errorSignal do: [:ex |
			self broadcast: 'Log Exception on "', SocketAccessor getHostname, '" -- ', ex errorString.
		]
	].
	^session!
!


!USPControllerService methodsFor: 'startup/shutdown'!
shutdown

	self stopResponder.
	outputSocket notNil ifTrue: [
		[
			runningHosts copy do: [:host |
				self removeHost: host].
			self broadcast: 'Shutting down ', SocketAccessor getHostname.
			(Delay forSeconds: 1) wait.  "Wait for message to be sent."
		] on: Object errorSignal do: [:ex |
			self
				at: 0 log: 'ERROR shutting down USPControllerService:';
				at: 0 log: '    "', ex errorString, "".
			ex return.
		].
		outputSocket close.
		outputSocket := nil].
	socket notNil ifTrue: [
		[
			socket close
		] on: Object errorSignal do: [:ex |
			self at: 0 log: 'ERROR closing responder socket'
		].
		socket := nil].!
!
```

```
!USPControllerService methodsFor: 'private - responder'!
startProcessId: anInteger
	"Start a consumer or producer process on this machine."

	| proc |
	proc := self processAt: anInteger.
	proc isNil ifTrue: [
		^self broadcast: 'Log Must install before starting'].
	proc isLocal ifFalse: [
		^self broadcast: 'Log Process start message was incorrectly routed'].
	proc start.!
startResponder

	[socket notNil] assert.
	self stopResponder.
	responder := Process
		forBlock: [self responderLoop]
		priority: Processor userSchedulingPriority + 1.  "Rapid response to short queries"
	responder resume.!
!


!USPControllerService methodsFor: 'startup/shutdown'!
startUp
	"Start me up.  Open a socket on my favorite port to listen for datagram messages."

	| addr |
	self shutdown.

	[socket isNil] assert.
	socket := SocketAccessor
		family: (SocketAddress domainCodeFromName: #aflnet)
		type: SocketAccessor sockDgram
		protocol: SocketAccessor pfUnspec.
	addr := IPSocketAddress hostName: SocketAccessor getHostname port: self class magicPort.
	[
		socket bindTo: addr.
	] on: OSErrorHolder existingReferentSignal do: [:ex |
		self at: 0 log: '*** PROBLEM: The magicPort (', self class magicPort printString, ') is currently in use.  Waiting 10s before retrying.'.
		(Delay forSeconds: 10) wait.
		ex retry
	].

	outputSocket notNil ifTrue: [
		[
			outputSocket shutdown: 2
		] on: Object errorSignal do: [:ex |
			"Ignore problems in shutdown: method."
			ex return.
		].
		outputSocket close].
	outputSocket := SocketAccessor
		family: (SocketAddress domainCodeFromName: #aflnet)
		type: SocketAccessor sockDgram
		protocol: SocketAccessor pfUnspec.
	addr := IPSocketAddress thisHostAnyPort.
	outputSocket bindTo: addr.

	self broadcast: self statusMessage.

	self startResponder.!
statusMessage

	| str localProcs |
	str := WriteStream on: (String new: 64).
	str nextPutAll: 'Running on '.
	str nextPutAll: SocketAccessor getHostname.
	localProcs := processes values select: [:proc |
		proc isLocal].
	(localProcs asSortedCollection: [:a :b | a myId <= b myId]) do: [:proc |
		str cr; print: proc myId; space; nextPutAll: proc statusString].
	^str contents!
!
```

```
!USPControllerService methodsFor: 'private - responder'!
stopProcessId: anInteger
	"Stop a consumer or producer process on this machine."

	| proc |
	proc := self processAt: anInteger.
	proc isNil ifTrue: [
		^self broadcast: 'Log Must start before stopping'].
	proc isLocal ifFalse: [
		^self broadcast: 'Log Process stop message was incorrectly routed'].
	proc stop.!
stopResponder

	| p |
	(p := responder) notNil ifTrue: [p terminate].
	responder := nil.!
syncTime: hostAndTimeString

	| str host timeString |
	str := hostAndTimeString readStream.
	host := str upTo: Character space.
	host = SocketAccessor getHostname ifTrue: [
		"Ignore loopback of broadcast, otherwise it'll drift my own clock a little each time."
		^self].
	timeString := str upToEnd.
	MiosoftWindowsSupport runDosAndWait: 'cmd /c time ', timeString.!
!


!USPControllerService methodsFor: 'accessing'!
syncToMe
	"Synchronize all clocks to the one on this processor (within about a second)."

	self broadcast: 'Sync to ', SocketAccessor getHostname, ' ', Time now printString!
!

-------------------------------------------------------------------------------

Smalltalk.Applications defineClass: #JobSchedulingFramework
	superclass: #{ENVY.Application}
	indexedType: #none
	private: false
	instanceVariableNames: ''
	classInstanceVariableNames: ''
	imports: ''
	category: ''!

Smalltalk defineClass: #UpdateStreamProcessor
	superclass: #{Core.Object}
	indexedType: #none
	private: false
	instanceVariableNames: '
		numContentionSpaces <uint16>
		numWriters <uint16>
		participants <ooVArray(UspParticipant)> '
	classInstanceVariableNames: ''
	imports: ''
	category: ''!
```

```
!UpdateStreamProcessor class methodsFor: 'database creation'!
createWithName: aString execution: executionCount writers: writerCount fractionBlock: fractionBlock
	"Create an UpdateStreamProcessor with the given information. The USP has a list of executionCount+writerCount
	 Participants, the first executionCount of which are mapped to contention spaces. The remainder are non-executing
	 participants, used for writing jobs into the USP. The USP will be written in the database 'USP <aString> root'. Each
	 participant will be written to a database named 'USP <aString> - <N>'."
	"UpdateStreamProcessor createWithName: 'Main' execution: 2 writers: 1 fractionBlock: [:x |]"

	^self new
		createWithName: aString
		execution: executionCount
		writers: writerCount
		fractionBlock: fractionBlock!
named: aString
	"Lookup and answer the UpdateStreamProcessor with the given name. Note that this is a transient
	 copy to prevent excessive locking. Call this within a transaction (preferrably MROW)."
	"DatabaseSession currentSession transactionMROW: [UpdateStreamProcessor named: 'Main']"

	^self named: aString inSession: DatabaseSession currentSession!
named: aString inSession: session
	"Lookup and answer the UpdateStreamProcessor with the given name. Note that this is a transient
	 copy to prevent excessive locking. Call this within a transaction (preferrably MROW)."
	"DatabaseSession currentSession transactionMROW: [UpdateStreamProcessor named: 'Main']"

	| db q |
	(session hasDB: 'USP ', aString, ' root') ifFalse: [^nil].
	db := session openDB: 'USP ', aString, ' root'.
	q := session
		lookupObj: 'USP'
		inScope: db.
	^q copy!
!


!UpdateStreamProcessor class methodsFor: 'private: generated'!
ooCodeGenVersion

	^ 1!
ooTypedInstanceVariablesString

	^ '
		numContentionSpaces <uint16>
		numWriters <uint16>
		participants <ooVArray(UspParticipant)> '!
!
```

```
!UpdateStreamProcessor methodsFor: 'initialize-release'!
createWithName: aString execution: executionCount writers: writerCount fractionBlock: fractionBlock
	"Create an UpdateStreamProcessor with the given information.  The USP has a list of executionCount+writerCount
	 Participants, the first executionCount of which are mapped to contention spaces.  The remainder are non-executing
	 participants, used for writing jobs into the USP.  The USP will be written in the database 'USP <aString> root'.  Each
	 participant will be written to a database named 'USP <aString> participant <N>'."

	| session total |
	numContentionSpaces := executionCount.
	numWriters := writerCount.
	total := writerCount + executionCount.
	participants := OoVArray new: total.
	session := DatabaseSession currentSession.
	session transaction: [
		| rootDB rootContainer |
		rootDB := session newDB: 'USP ', aString, ' root'.
		rootContainer := rootDB newCPPContainer: 'USP Root container'.
		rootContainer cluster: self.
		fractionBlock value: 1 / (total + 1).

		1 to: total do: [:count |
			| isContentionSpace participant participantDB participantCon |
			isContentionSpace := count <= executionCount.
			isContentionSpace
				ifTrue: [
					participant := ContentionSpace new
						initializeId: count
						participants: total
						consumers: numContentionSpaces
						hostName: ''
						port: 13794]
				ifFalse: [
					participant := UspParticipant new
						initializeId: count
						participants: total
						consumers: numContentionSpaces
						hostName: ''].
			participantDB := session newDB: 'USP ', aString, ' - ', count printString.
			participantCon := participantDB newCPPContainer: 'USP Participant container'.
			participantCon cluster: participant.
			participants at: count put: participant.
			fractionBlock value: (count + 1) / (total + 1)].

		"Name the scheduler."
		session
			nameObj: self
			with: 'USP'
			inScope: rootDB.
	].
	^self!
!


!UpdateStreamProcessor methodsFor: 'accessing'!
numberOfContentionSpaces

	^numContentionSpaces!
numberOfParticipants

	^numContentionSpaces + numWriters!
participantAt: index

	^participants at: index!
!


!UpdateStreamProcessor methodsFor: 'copying'!
postCopy

	super postCopy.
	participants := participants copy.!
!
```

```
Smalltalk defineClass: #UspParticipant
	superclass: #{Core.Object}
	indexedType: #none
	private: false
	instanceVariableNames: '
		id <uint16>
		hostName <ooVString>
		outputBuffers <ooVArray(ooShortRef(JobBuffer))>
		uniquenessCounter <uint64>
		numberOfParticipants <uint16> '
	classInstanceVariableNames: ''
	imports: ''
	category: ''!


!UspParticipant class methodsFor: 'private: generated'!
ooCodeGenVersion

	^ 1!
ooTypedInstanceVariablesString

	^ '
		id <uint16>
		hostName <ooVString>
		outputBuffers <ooVArray(ooShortRef(JobBuffer))>
		uniquenessCounter <uint64>
		numberOfParticipants <uint16> '!
!


!UspParticipant methodsFor: 'ui support'!
assignmentString

	| str |
	str := WriteStream on: (String new: 64).
	str nextPutAll: 'Currently assigned to '.
	str nextPutAll: (hostName isNil ifTrue: ['(nil)'] ifFalse: [hostName]).
	^str contents!
dumpStatusOn: str

	str nextPutAll: 'Pro#'; print: id.!
!


!UspParticipant methodsFor: 'accessing'!
hostName

	^hostName!
!


!UspParticipant methodsFor: 'initialize-release'!
initializeId: myId participants: totalCount consumers: consumers hostName: myHost

	id := myId.
	outputBuffers := OoVArray new: consumers.
	1 to: consumers do: [:i |
		outputBuffers at: i put: self jobBufferClass new].
	uniquenessCounter := myId.
	numberOfParticipants := totalCount.
	hostName := myHost.!
!


!UspParticipant methodsFor: 'accessing'!
jobBufferClass

	self ooClass == UspParticipant ifTrue: [^SaturationTestJobBuffer].
	^JobBuffer!
myId

	^id!
numberOfContentionSpaces

	^outputBuffers size!
```

```
numberOfParticipants

    ^numberOfParticipants!
outputBuffers

    ^outputBuffers!
!


!UspParticipant methodsFor: 'ui support'!
resetAllJobs
    "This is extremely dangerous, and is only to be used for testing.  Remove all jobs, and reset
     the job numbering of my output buffers."

    outputBuffers do: [:buffer |
        buffer resetAllJobs].!
!


!UspParticipant methodsFor: 'accessing'!
setHostName: newHostName

    hostName := newHostName.!
!


!UspParticipant methodsFor: 'ui support'!
statusString

    | str |
    str := WriteStream on: (String new: 64).
    self dumpStatusOn: str.
    ^str contents!
!

Smalltalk defineClass: #ContentionSpace
    superclass: #{UspParticipant}
    indexedType: #none
    private: false
    instanceVariableNames: '
        port <uint16>
        lastJobsExecuted <ooVArray(uint32)>
        runningJob <ooShortRef(Job)>
        runningTagInteger <uint64>
        waitingSyncJobs <ooShortRef(OrderedCollection)> '
    classInstanceVariableNames: ''
    imports: ''
    category: ''!


!ContentionSpace class methodsFor: 'private: generated'!
ooCodeGenVersion

    ^ 1!
ooTypedInstanceVariablesString

    ^'
        port <uint16>
        lastJobsExecuted <ooVArray(uint32)>
        runningJob <ooShortRef(Job)>
        runningTagInteger <uint64>
        waitingSyncJobs <ooShortRef(OrderedCollection)> '!
!
```

```
!ContentionSpace methodsFor: 'ui support'!
assignmentString

	| str |
	str := WriteStream on: (String new: 64).
	str nextPutAll: 'Currently assigned to '.
	str nextPutAll: hostName, '@', port printString.
	^str contents!
dumpStatusOn: str

	str nextPutAll: 'Con#'; print: id.
	str nextPutAll: ', host='; nextPutAll: hostName.
	str nextPutAll: ', port='; print: port.
	str nextPutAll: '.'.!
!


!ContentionSpace methodsFor: 'initialize-release'!
initializeId: myId participants: totalCount consumers: consumers hostName: myHost port: myPort

	self initializeId: myId participants: totalCount consumers: consumers hostName: myHost.

	port := myPort.
	lastJobsExecuted := OoVArray new: totalCount withAll: 0.
	runningJob := nil.
	runningTagInteger := 0.
	waitingSyncJobs := OrderedCollection new.!
!


!ContentionSpace methodsFor: 'accessing'!
lastJobsExecuted

	^lastJobsExecuted!
nextUniqueInteger
	"Answer an integer that is unique for this row. Use a 64-bit counter, which will
	repeat every 18 million million million elements or so. At a billion allocations a second
	(a ridiculous overestimate), this would still last > 580 years before repeating. Of
	course, we should temper this with the fact that every row and column has a
	unique modulus in this huge set. Thus, if there are 1000 rows and 3000 columns
	(a very large scale application, indeed), the integers would cycle approximately
	every 580/4000.0 = ~1/7th of a year. Of course, the combined computation would
	have to be exceeding four trillion allocations a second!! Anyhow, as long as old
	jobs get processed before these counters wrap around -- no problem. Note that
	this method dirties this ContentionSpace, so it should only be used when the
	ContentionSpace is already write-locked."

	uniquenessCounter := uniquenessCounter + numberOfParticipants.
	uniquenessCounter basicSize > 8 ifTrue: [
		"8 bytes = 64 bits."
		"Avoid ever returning zero, as that is used to indicate unsynchronized Job execution."
		uniquenessCounter := (uniquenessCounter - 1) \\ numberOfParticipants + 1].
	^uniquenessCounter!
port

	^port!
!
```

```
!ContentionSpace methodsFor: 'ui support'!
resetAllJobs
	"This is extremely dangerous, and is only to be used for testing.  Remove all jobs, and reset
	 the job numbering of my output buffers."

	super resetAllJobs.

	[waitingSyncJobs isEmpty] whileFalse: [
		waitingSyncJobs removeFirst ooDelete].

	runningJob == nil ifFalse: [
		runningJob ooDelete.
		runningJob := nil].

	lastJobsExecuted atAllPut: 0.

	runningTagInteger := 0.!
!


!ContentionSpace methodsFor: 'accessing'!
runningJob

	^runningJob!
runningJob: aJob

	runningJob := aJob.!
runningTag
	"Answer a (lightweight) JobSynchronizationTag that is used to identify jobs that
	 must be synchronized together.  Answer nil if a synchronized group is not active."

	runningTagInteger == 0 ifTrue: [^nil].
	^JobSynchronizationTag new fromInteger: runningTagInteger!
runningTag: aJobSynchronizationTagOrNil
	"Set my curren tag.  This indicates that in the case of recovery we should execute
	 jobs that have this tag right after the currentJob (if any) completes."

	aJobSynchronizationTagOrNil isNil
		ifTrue: [runningTagInteger := 0]
		ifFalse: [runningTagInteger := aJobSynchronizationTagOrNil tagInteger]!
setHostName: newHostName port: newPort

	hostName := newHostName.
	port := newPort.!
waitingSyncJobs

	^waitingSyncJobs!
!
```

```
Smalltalk defineClass: #Query
	superclass: #{Formula}
	indexedType: #none
	private: false
	instanceVariableNames: '
		autoCreate <boolean>
		index <Index>'
	classInstanceVariableNames: ''
	imports: ''
	category: 'ENVY/Manager'!


!Query class methodsFor: 'private: generated'!
ooCodeGenVersion

	^ 1!
ooTypedInstanceVariablesString

	^ '
		autoCreate <boolean>
		index <Index>'!
!


Smalltalk defineClass: #IndexEntry
	superclass: #{Core.Object}
	indexedType: #none
	private: false
	instanceVariableNames: '
		indexedObject <Object> '
	classInstanceVariableNames: 'cachedIndex '
	imports: ''
	category: 'ENVY/Manager'!


!IndexEntry class methodsFor: 'class accessing'!
cacheIndexIfNilForSession: aSession
	"Cache my index if it's nil.  Must be called within an MROW transaction of aSession, otherwise
	 it might lock the System container, which could be a Bad Thing."

	| schema cls ind |
	cachedIndex notNil ifTrue: [^cachedIndex].
	schema := (MioSystem currentPersistent: aSession) schema.
	cls := schema schemaClassNamed: self rootClassName.
	ind := cls indices
		detect: [:eachInd | eachInd name = self indexName]
		ifNone: [self error: 'The index for this IndexEntry no longer exists in the schema'].
	cachedIndex := ind deepCopyCreatingHomomorphism: IdentityDictionary new.
	^cachedIndex!
!


!IndexEntry class methodsFor: 'class initialization'!
clearCachedIndices

	cachedIndex := nil.!
!


!IndexEntry class methodsFor: 'class accessing'!
index
	"Answer the instance of Index for which I represent an entry.  The cache
	 must have already been set up via cacheIndexIfNilForSession: first."

	cachedIndex isNil ifTrue: [
		self error: 'Make sure to call cacheIndexIfNilForSession: first.'].
	^cachedIndex!
indexName
	"Answer the name of the Index for which I represent instances."

	self ooIsPersistent ifTrue: [^superclass indexName].
	^self fullName readStream upTo: $_; upToEnd!
!
```

```
!IndexEntry class methodsFor: 'class initialization'!
initialize

	self withAllSubclassesDo: [:cls |
		cls clearCachedIndices].!
!


!IndexEntry class methodsFor: 'instance creation'!
new

	^super new initialize!
!


!IndexEntry class methodsFor: 'private: generated'!
ooCodeGenVersion

	^ 1!
ooTypedInstanceVariablesString

	^'
		indexedObject <Object> '!
!


!IndexEntry class methodsFor: 'class accessing'!
rootClassName
	"Answer the name of the root class for which I represent instances."

	self ooIsPersistent ifTrue: [^superclass rootClassName].
	^self fullName readStream upTo: $_!
!


!IndexEntry methodsFor: 'accessing'!
contentionIndex
	"Answer the contention space that I should be in.  Since this was assigned randomly to the index's
	containers during index creation, we must defer to the index to figure this out."

	^self ooClass index contentionSpaceFor: self!
!


!IndexEntry methodsFor: 'comparing'!
hashOfKey
	"Answer a hash of just my key information."

	self subclassResponsibility.!
hasSameKeyAs: another
	"Answer whether my key information matches another's key information."

	[self ooClass == another ooClass] assert.
	self subclassResponsibility.!
!


!IndexEntry methodsFor: 'accessing'!
indexedObject

	^ indexedObject!
indexedObject: anObject

	indexedObject := anObject.!
!


!IndexEntry methodsFor: 'initialize-release'!
initialize
	"Subclasses will do more."

	^self!
!
```

```
!IndexEntry methodsFor: 'accessing'!
key

	^IndexEntryKey fromIndexEntry: self!
targetContainerIn: session
	"Answer the OoContainer in which to insert this IndexEntry."

	^((self ooClass cacheIndexIfNilForSession: session)
			containerFor: self inSession: session
		) ooContainer!
!

Smalltalk defineClass: #IndexEntryKey
	superclass: #{Core.Object}
	indexedType: #none
	private: false
	instanceVariableNames: 'indexEntry '
	classInstanceVariableNames: ''
	imports: ''
	category: 'ENVY/Manager'!


!IndexEntryKey class methodsFor: 'instance creation'!
fromIndexEntry: anIndexEntry

	^self new indexEntry: anIndexEntry!
!


!IndexEntryKey methodsFor: 'comparing'!
= another
	"Compare this index entry key to the other one."

	another assertType: IndexEntryKey.  "Safety precaution for now"
	indexEntry ooClass = another indexEntry ooClass ifFalse: [^false].  "Different kinds of index entries"
	^indexEntry hasSameKeyAs: another indexEntry!
hash
	"Answer this key's hash value."

	^indexEntry hashOfKey!
!


!IndexEntryKey methodsFor: 'accessing'!
indexEntry

	^indexEntry!
!


!IndexEntryKey methodsFor: 'initialization'!
indexEntry: anIndexEntry

	indexEntry := anIndexEntry.!
!


Smalltalk defineClass: #IndexEntryTracker
	superclass: #{Core.Object}
	indexedType: #none
	private: false
	instanceVariableNames: ''
	classInstanceVariableNames: ''
	imports: ''
	category: 'ENVY/Manager'!
```

```
!IndexEntryTracker methodsFor: 'accessing'!
checkForPendingRequestsForObject: object contentionSpaceProcess: aContentionSpaceProcess contentionIndex: contentionIndex

    [object isStable not] assert.
    object isDeleting ifTrue: [
        "I've just finished deleting all my index entries in preparation for my own deletion.
         Since I have already long since disconnected my change node and sent it a reply
         indicating this fact, I should not send it any message (I can't anyhow, since my
         pointer to it was already cleared)."
        [object localIndexEntries isEmpty & object remoteIndexEntries isEmpty] assert.
        object delete.
        ^self].

    [object isReindexing] assert.
    object beStable.
    object hasRequestedDeletion ifTrue: [
        | tag remoteEntries localEntries |
        object clearRequests; beDeleting.
        tag := aContentionSpaceProcess nextUniqueInteger.
        remoteEntries := object remoteIndexEntries.
        remoteEntries isEmpty
            ifTrue: [object delete]
            ifFalse: [
                localEntries := object localIndexEntries.
                [localEntries size = remoteEntries size] assert.
                localEntries with: remoteEntries do: [:localEntry :remoteEntry |
                    localEntry ooClass cacheIndexIfNilForSession: aContentionSpaceProcess session.
                    aContentionSpaceProcess addJob: (IndexEntryDeleteJob new
                        contentionIndex: localEntry contentionIndex;
                        indexedObject: object;
                        indexEntry: remoteEntry;
                        replyTag: tag;
                        replyQuorumDenominator: localEntries size;
                        objectContentionIndex: contentionIndex).
                    localEntry ooDelete].
                self localIndexEntries: Array new.
                self remoteIndexEntries: Array new].
        ^self].

    object hasRequestedReindexing ifTrue: [
        "An indexing was requested while my previous indexing was being done.  Start reindexing once again."
        [object localIndexEntries size = object remoteIndexEntries size] assert.
        object clearRequests.
        self
            contentionSpaceProcess: aContentionSpaceProcess
            object: object
            objectContentionIndex: contentionIndex
            oldTransientsIfKnown: object localIndexEntries.
        ^self].

    "The object just became stable.  Delete its local index entries, as these can now be computed from the object itself."
    object localIndexEntries do: [:ie | ie ooDelete].
    object localIndexEntries: Array new.!
```

```
contentionSpaceProcess: aContentionSpaceProcess object: object objectContentionIndex: objectContentionIndex oldTransientsIfKnown: oldTransientsOrNil
	"The old object has just been replaced with the new object.  Create index entry update jobs.  Note that
	 the old and new object may share parts, and the old object may have been partially deleted already
	 at this point.  If the list of oldTransients (IndexEntries) is provided, it is assumed that these were extracted
	 from the old object before it was destroyed.  If no oldTransients are provided, they must be present in the
	 localIndexEntries instance variable.  We must be in a transaction.  If any jobs are scheduled (which will
	 eventually reply to the object), set the object's isReindexing flag."

	| oldTransients oldTransientsToSubscripts newTransients remoteEntries newMixed oldKeysToSubscripts typesAndContentionToSubscripts quorum
resultingEntries replyTag |
	[object isStable] assert.

	remoteEntries := object remoteIndexEntries.
	oldTransientsOrNil isNil
		ifTrue: [
			"This means the object was already changed back when the object's indexes were in the
			 process of being updated (for a *previous* change).  Now we're playing catch-up, but the
			 local and remote index entries will still agree.  Therefore, use the local entries to figure out
			 which remote entries need to be updated in which ways."
			oldTransients := object localIndexEntries]
		ifFalse: [
			"In this case the object has not changed since the previous indexing *started*.  The passed
			 list of transient index entries should agree with the current persistent entries.  This will save
			 us having to actually read the remote persistent index entries in this process."
			oldTransients := oldTransientsOrNil].

	[oldTransients size = remoteEntries size] assert: 'The object''s index entries are invalid'.
	"At this point oldTransients agrees structurally with remoteEntries, the list of current persistent IndexEntries
	 connected to the object."

	oldTransientsToSubscripts := Dictionary new.
	1 to: oldTransients size do: [:i |
		| list |
		list := oldTransientsToSubscripts at: (oldTransients at: i) ifAbsentPut: [OrderedCollection new].
		list add: i].

	"First try to reuse persistent index entries that exactly match the new ones..."
	newTransients := object transientIndexEntries.
	newMixed := newTransients collect: [:newTrans |
		| subs |
		subs := oldTransientsToSubscripts at: newTrans ifAbsent: [#()].
		subs notEmpty
			ifTrue: [
				"Use an existing persistent one that's just like it."
				#oldUnchanged -> (remoteEntries at: subs removeFirst)]
			ifFalse: [
				#maybeNew -> newTrans]].

	"In preparation for what follows, ensure all Index information has been cached.  This just includes
	 information about which hash value goes to which index container, and what contention space it's in."
	oldTransients do: [:trans |
		trans ooClass cacheIndexIfNilForSession: aContentionSpaceProcess session].
	newTransients do: [:trans |
		trans ooClass cacheIndexIfNilForSession: aContentionSpaceProcess session].

	"Now try to reuse unclaimed persistent index entries that have the same keys as new ones..."
	oldKeysToSubscripts := Dictionary new.
	oldTransientsToSubscripts keysAndValuesDo: [:oldTrans :subs |
		subs size > 0 ifTrue: [
			| list |
			list := oldKeysToSubscripts at: oldTrans key ifAbsentPut: [OrderedCollection new].
			list addAll: subs]].
	newMixed := newMixed collect: [:newAssoc |
		newAssoc key = #maybeNew
			ifTrue: [
				| newTrans newTransKey subs |
				newTrans := newAssoc value.
				newTransKey := newTrans key.
				subs := oldKeysToSubscripts at: newTransKey ifAbsent: [#()].
				subs notEmpty
					ifTrue: [
						| sub |
						sub := subs removeFirst.
						#oldChanged -> (Array with: (remoteEntries at: sub) with: (oldTransients at: sub) with: newTrans)]
					ifFalse: [
						newAssoc]]
```

```
            ifFalse: [newAssoc]].

"As a final optimization, attempt to recycle index entries with merely the correct ooClass and contention space..."
typesAndContentionToSubscripts := Dictionary new.
oldKeysToSubscripts keysAndValuesDo: [:newTransKey :subs |
      subs do: [:sub |
            | contention pair list |
            contention := (oldTransients at: sub) contentionIndex.
            pair := Array with: newTransKey indexEntry ooClass with: contention.
            list := typesAndContentionToSubscripts at: pair ifAbsentPut: [OrderedCollection new].
            list add: sub]].
newMixed := newMixed collect: [:newAssoc |
      newAssoc key = #maybeNew
            ifTrue: [
                  | newTrans contention pair subs |
                  newTrans := newAssoc value.
                  contention := newTrans contentionIndex.
                  pair := Array with: newTrans ooClass with: contention.
                  subs := typesAndContentionToSubscripts at: pair ifAbsent: [#()].
                  subs notEmpty
                        ifTrue: [
                              | sub |
                              sub := subs removeFirst.
                              #oldChanged -> (Array with: (remoteEntries at: sub) with: (oldTransients at: sub) with: newTrans)]
                        ifFalse: [
                              #newEntry -> (Array with: newTrans)]]
            ifFalse: [newAssoc]].

"Now create the actual jobs to carry out this plan.  Start by deleting unrecycleable IndexEntries..."
quorum := (newMixed select: [:assoc | assoc key ~= #oldUnchanged]) size.
typesAndContentionToSubscripts do: [:subs |
      quorum := quorum + subs size].
quorum > 0 ifTrue: [
      object beReindexing.
      replyTag := aContentionSpaceProcess nextUniqueInteger].
typesAndContentionToSubscripts keysAndValuesDo: [:pair :subs |
      subs do: [:sub |
            | job |
            job := IndexEntryDeleteJob new
                  contentionIndex: pair last;
                  indexedObject: object;
                  indexEntry: (remoteEntries at: sub);
                  replyTag: replyTag;
                  replyQuorumDenominator: quorum;
                  objectContentionIndex: objectContentionIndex.
            aContentionSpaceProcess addJob: job]].

"Deal with the new and recycled index entries..."
resultingEntries := OrderedCollection new: newMixed size.
1 to: newMixed size do: [:subscript |
      | newAssoc value |
      newAssoc := newMixed at: subscript.
      value := newAssoc value.
      (Case of: newAssoc key)
            if: [#oldUnchanged] do: [
                  "Do nothing - the index entry is still valid..."
                  resultingEntries add: value];
            if: [#oldChanged] do: [
                  "The entry's contention space is the same, but its data has changed..."
                  | oldPersist oldTrans new job |
                  oldPersist := value at: 1.
                  oldTrans := value at: 2.
                  new := value at: 3.
                  object ooCluster: new.  "Until it's deleted in a reply job."
                  job := IndexEntryUpdateJob new
                        contentionIndex: oldTrans contentionIndex;
                        indexEntryToUpdate: oldPersist;
                        newIndexEntryData: new;
                        replyContentionIndex: objectContentionIndex;
                        replyQuorumDenominator: quorum;
                        replyTag: replyTag.
                  aContentionSpaceProcess addJob: job.
                  resultingEntries add: oldPersist];
            if: [#newEntry] do: [
                  "Create a new index entry..."
                  | new job |
```

```
                    new := value first.
                    object ooCluster: new.  "Until it's deleted in a reply job."
                    job := IndexEntryAddJob new
                        indexEntry: value first;
                        subscript: subscript;
                        replyContentionIndex: objectContentionIndex;
                        replyQuorumDenominator: quorum;
                        replyTag: replyTag.
                    aContentionSpaceProcess addJob: job.
                    "Since the actual index entry does not yet exist, just refer to the temporary one."
                    resultingEntries add: nil];
                else: [self error: 'Unrecognized index entry update mode']].
    object remoteIndexEntries: resultingEntries.
    oldTransients do: [:ie |
        ie ooIsPersistent ifTrue: [ie ooDelete]].
    object localIndexEntries: newTransients.!
!

Smalltalk defineClass: #IndexObserver
    superclass: #{Core.Object}
    indexedType: #none
    private: false
    instanceVariableNames: ''
    classInstanceVariableNames: ''
    imports: ''
    category: 'ENVY/Manager'!


!IndexObserver methodsFor: 'notification'!
index: anIndex hasAdded: anIndexEntry
    self implementedBySubclass!
!

Smalltalk defineClass: #AbstractSetFieldJob
    superclass: #{OneStepJob}
    indexedType: #none
    private: false
    instanceVariableNames: '
        object <Object>
        rootObject <Object>
        fieldName <ooVString> '
    classInstanceVariableNames: ''
    imports: ''
    category: ''!


!AbstractSetFieldJob class methodsFor: 'private: generated'!
ooCodeGenVersion

    ^ 1!
ooTypedInstanceVariablesString

    ^ '
        object <Object>
        rootObject <Object>
        fieldName <ooVString> '!
!
```

```
!AbstractSetFieldJob methodsFor: 'execution'!
executeWith: aContentionSpaceProcess
    "Make the required change to object, causing rootObject to be reindexed (as necessary)."

    rootObject isStable
        ifTrue: [
            | oldTransients |
            oldTransients := rootObject transientIndexEntries.
            object perform: fieldName with: self value.
            IndexEntryTracker new
                contentionSpaceProcess: aContentionSpaceProcess
                object: rootObject
                objectContentionIndex: aContentionSpaceProcess myId
                oldTransientsIfKnown: oldTransients]
        ifFalse: [
            rootObject isDeleting ifTrue: [^self].  "Ignore changes to doomed objects."
            [rootObject isReindexing] assert.  "not stable means either deleting or reindexing."
            "The reindexing will happen after the current reindexing completes."
            object perform: fieldName with: self value.
            rootObject requestReindexing].!
!


!AbstractSetFieldJob methodsFor: 'accessing'!
fieldName: aSymbol

    aSymbol assertType: Symbol.
    aSymbol last = $: ifFalse: [
        self error: 'The fieldName must be a one-argument selector'].
    fieldName := aSymbol.!
object: anObject

    anObject class ooIsPersistent ifFalse: [
        self error: 'The object must already have been clustered (or persistent)'].
    object := anObject.!
!


!AbstractSetFieldJob methodsFor: 'oo-persistence'!
ooInitializeAfterRead
    "Sent immediately after the receiver is read from the database. You can override
    this method to initialize the receiver after reading it. The object returned by
    this method is ignored."

    fieldName := fieldName asSymbol.
    ^super ooInitializeAfterRead!
!


!AbstractSetFieldJob methodsFor: 'accessing'!
rootObject: anObject

    anObject class ooIsPersistent ifFalse: [
        self error: 'The rootObject must already have been clustered (or persistent)'].
    rootObject := anObject.!
!


Smalltalk defineClass: #SetFieldToOidJob
    superclass: #{AbstractSetFieldJob}
    indexedType: #none
    private: false
    instanceVariableNames: '
        value <Object> '
    classInstanceVariableNames: ''
    imports: ''
    category: ''!
```

```
!SetFieldToOidJob class methodsFor: 'private: generated'!
ooCodeGenVersion

    ^ 1!
ooTypedInstanceVariablesString

    ^ '
        value <Object> '!
!


!SetFieldToOidJob methodsFor: 'accessing'!
value

    ^value!
value: aPersistentObject

    aPersistentObject class ooIsPersistent ifFalse: [
        self error: 'This object must already be clustered (or fully persistent)'].
    value := aPersistentObject.!
!

Smalltalk defineClass: #SetFieldToStringJob
    superclass: #{AbstractSetFieldJob}
    indexedType: #none
    private: false
    instanceVariableNames: '
        value <ooVString> '
    classInstanceVariableNames: ''
    imports: ''
    category: ''!


!SetFieldToStringJob class methodsFor: 'private: generated'!
ooCodeGenVersion

    ^ 1!
ooTypedInstanceVariablesString

    ^ '
        value <ooVString> '!
!


!SetFieldToStringJob methodsFor: 'accessing'!
value

    ^value!
value: aString

    aString assertType: String.
    value := aString.!
!

Smalltalk defineClass: #SetFieldToValueJob
    superclass: #{AbstractSetFieldJob}
    indexedType: #none
    private: false
    instanceVariableNames: '
        value <ooShortRef(Object)> '
    classInstanceVariableNames: ''
    imports: ''
    category: ''!


!SetFieldToValueJob class methodsFor: 'private: generated'!
ooCodeGenVersion

    ^ 1!
ooTypedInstanceVariablesString

    ^ '
        value <ooShortRef(Object)> '!
```

```
!

!SetFieldToValueJob methodsFor: 'deleting'!
ooDelete

	value ooDelete.
	super ooDelete.!
!


!SetFieldToValueJob methodsFor: 'copying'!
postCopy

	super postCopy.
	value := value copy.!
!


!SetFieldToValueJob methodsFor: 'accessing'!
value

	^value!
value: anObject
	"This will be sent *by value* to the contention space that will execute it."

	value := anObject.!
!

Smalltalk defineClass: #CreateFakeObjectsJob
	superclass: #{OneStepJob}
	indexedType: #none
	private: false
	instanceVariableNames: '
		count <uint32> '
	classInstanceVariableNames: ''
	imports: ''
	category: ''!


!CreateFakeObjectsJob class methodsFor: 'class accessing'!
addRootObject: aRootObject
	"Add the root object to my collection of all root objects."

	AllRootObjects addLast: aRootObject.!
clearAllRoots
	"Empty my collection of root objects."

	[AllRootObjects isEmpty] whileFalse: [AllRootObjects removeFirst].!
!


!CreateFakeObjectsJob class methodsFor: 'class initialization'!
initialize

	AllRootObjects := OrderedCollection new.!
!


!CreateFakeObjectsJob class methodsFor: 'private: generated'!
ooCodeGenVersion

	^ 1!
ooTypedInstanceVariablesString

	^'
		count <uint32> '!
!
```

```
!CreateFakeObjectsJob class methodsFor: 'class accessing'!
pickProductUsing: subscript contentionSpaceProcess: aContentionSpaceProcess

	| prod con obj |
	AllRootObjects size = 0 ifTrue: [
		self recacheRootObjectsFor: aContentionSpaceProcess].
	prod := AllRootObjects at: subscript \\ AllRootObjects size + 1.
	con := aContentionSpaceProcess session
		ooProvideContainerFor: prod ooContainer ooPrivateContainer ooContainerNumber
		with: prod ooObjectNumber.
	obj := con ooProvideObject: prod ooObjectNumber.
	^obj!
randomString

	| str rnd |
	rnd := BouncingJob random.
	str := String new: (rnd next * 10) truncated + 5.
	str at: 1 put: (Character value: (rnd next * 26) truncated + $A asInteger).
	2 to: str size do: [:i |
		str at: i put: (Character value: (rnd next * 26) truncated + $a asInteger)].
	^str!
recacheRootObjectsFor: aContentionSpaceProcess
	"Regenerate my cached collection of products from the database."

	| session dbName db scan |
	session := aContentionSpaceProcess session.
	self clearAllRoots.
	dbName := 'Telecom_Person - ', aContentionSpaceProcess myId printString.
	db := session openDB: dbName.
	scan := db scan: Telecom_Person.
	[scan atEnd] whileFalse: [
		self addRootObject: scan next].!
!


!CreateFakeObjectsJob class methodsFor: 'EM-Internal'!
_PRAGMA_

	"(defineStatic: #AllRootObjects private: false constant: false category: 'As yet unclassified' initializer: nil)"!
!


!CreateFakeObjectsJob methodsFor: 'accessing'!
count: anInteger

	count := anInteger.!
!
```

```
!CreateFakeObjectsJob methodsFor: 'execution'!
executeWith: aContentionSpaceProcess
	"Create the required number of Products, filled with random strings and numbers."

	| session dbName db con |
	session := aContentionSpaceProcess session.
	dbName := 'Telecom_Person - ', aContentionSpaceProcess myId printString.
	(session hasDB: dbName)
		ifTrue: [db := session openDB: dbName]
		ifFalse: [
			| systemName path |
			systemName := session systemName.
			path := MioSystem defaultDatabasePath, dbName, '.', systemName, '.DB'.
			db := session
				newDB: dbName
				defaultContPages: 0
				growth: 0
				host: SocketAccessor getHostname
				path: path].
	(db hasContainer: dbName)
		ifTrue: [con := db openContainer: dbName]
		ifFalse: [con := db newContainer: dbName].
	count timesRepeat: [
		| person name addr acct billAddr loc line plan |
		person := Telecom_Person new.
		name := Telecom_Name new.
		acct := Telecom_Account new.
		addr := Telecom_Address new.
		billAddr := Telecom_Address new.
		loc := Telecom_Address new.
		line := Telecom_Line new.
		plan := Telecom_FlatRatePlan new.
		person
			birthDate: Date today;
			email: 'nobody@there.com';
			ssn: (BouncingJob random next * 499999999.0) truncated + 1;
			addAddress: addr;
			name: name;
			addAccount: acct.
		name
			first: self class randomString;
			middle: self class randomString;
			last: self class randomString.
		acct
			accountNumber: (BouncingJob random next * 999999.0) truncated + 1;
			addLine: line;
			billingAddress: billAddr;
			billingMethod: Telecom_MonthlyInvoice new.
		addr
			country: self class randomString;
			state: self class randomString;
			city: self class randomString;
			street: self class randomString;
			postalCode: self class randomString.
		billAddr
			country: self class randomString;
			state: self class randomString;
			city: self class randomString;
			street: self class randomString;
			postalCode: self class randomString.
		loc
			country: self class randomString;
			state: self class randomString;
			city: self class randomString;
			street: self class randomString;
			postalCode: self class randomString.
		line
			number: (BouncingJob random next * 9999999999.0) truncated + 1;
			addPlan: plan;
			location: loc.
		person clusterWith: con.
		self class addRootObject: person.   "For the test harness"
		"Send jobs to create index entries for this new product."
		IndexEntryTracker new
```

```
                contentionSpaceProcess: aContentionSpaceProcess
                object: person
                objectContentionIndex: contentionIndex
                oldTransientsIfKnown: #().
        ].!
!

Smalltalk defineClass: #DeleteProductsJob
        superclass: #{OneStepJob}
        indexedType: #none
        private: false
        instanceVariableNames: ''
        classInstanceVariableNames: ''
        imports: ''
        category: ''!


!DeleteProductsJob methodsFor: 'execution'!
executeWith: aContentionSpaceProcess
        "Delete all Products from this contention space."

        | session dbName db scan |
        session := aContentionSpaceProcess session.
        dbName := 'Telecom_Person - ', aContentionSpaceProcess myId printString.
        db := session openDB: dbName.
        scan := db scan: Telecom_Person.
        [scan atEnd] whileFalse: [
                | prod localEntries remoteEntries |
                prod := scan next.
                localEntries := prod localOrTransientIndexEntries.
                remoteEntries := prod remoteIndexEntries.
                prod isStable
                        ifTrue: [
                                [localEntries size = remoteEntries size] assert.
                                remoteEntries isEmpty
                                        ifTrue: [prod delete]
                                        ifFalse: [
                                                | tag |
                                                tag := aContentionSpaceProcess nextUniqueInteger.
                                                localEntries with: remoteEntries do: [:localEntry :remoteEntry |
                                                        localEntry ooClass cacheIndexIfNilForSession: aContentionSpaceProcess session.
                                                        aContentionSpaceProcess addJob: (IndexEntryDeleteJob new
                                                                contentionIndex: localEntry contentionIndex;
                                                                indexedObject: prod;
                                                                indexEntry: remoteEntry;
                                                                replyTag: tag;
                                                                replyQuorumDenominator: localEntries size;
                                                                objectContentionIndex: contentionIndex)].
                                                prod beDeleting.
                                                prod localIndexEntries: Array new.
                                                prod remoteIndexEntries: Array new]]
                                ifFalse: [
                                        prod isDeleting ifFalse: [
                                                prod requestDeletion]].
        ].
        CreateFakeObjectsJob clearAllRoots.!
!

Smalltalk defineClass: #EstablishMultipleLinksToIndexEntriesWithCleanupJob
        superclass: #{OneStepJob}
        indexedType: #none
        private: false
        instanceVariableNames: '
                indexedObject <Object>
                subscripts <ooVArray(uint32)>
                indexEntries <ooVArray(IndexEntry)>
                tempIndexEntriesToDelete <ooVArray(IndexEntry)> '
        classInstanceVariableNames: ''
        imports: ''
        category: 'ENVY/Manager'!
```

```
!EstablishMultipleLinksToIndexEntriesWithCleanupJob class methodsFor: 'private: generated'!
ooCodeGenVersion

    ^ 1!
ooTypedInstanceVariablesString

    ^ '
        indexedObject <Object>
        subscripts <ooVArray(uint32)>
        indexEntries <ooVArray(IndexEntry)>
        tempIndexEntriesToDelete <ooVArray(IndexEntry)> '!


!EstablishMultipleLinksToIndexEntriesWithCleanupJob methodsFor: 'accessing'!
addIndexEntryToDelete: tempIndexEntry

    tempIndexEntriesToDelete addElement: tempIndexEntry.!
addSubscript: subscript newIndexEntry: newIndexEntry

    subscripts addElement: subscript.
    indexEntries addElement: newIndexEntry.!


!EstablishMultipleLinksToIndexEntriesWithCleanupJob methodsFor: 'execution'!
executeWith: aContentionSpaceProcess
    "Update the given subscripts of my indexed object's list of index entries to point
     to the freshly constructed index entries.  Also delete the temporary index entries.
     Also start a reindex or delete operation based on the object's requestFlag."

    [subscripts size = indexEntries size] assert.

    subscripts with: indexEntries do: [:i :entry |
        indexedObject remoteIndexEntries at: i put: entry].

"   tempIndexEntriesToDelete do: [:entry | entry ooDelete]."   "Ignore - I've changed this mechanism"
    [tempIndexEntriesToDelete all: [:entry |
        indexedObject localIndexEntries any: [:localEntry | localEntry == entry]].
    ] assert.

    IndexEntryTracker new
        checkForPendingRequestsForObject: indexedObject
        contentionSpaceProcess: aContentionSpaceProcess
        contentionIndex: contentionIndex.!


!EstablishMultipleLinksToIndexEntriesWithCleanupJob methodsFor: 'accessing'!
indexedObject: anObject

    indexedObject := anObject.!


!EstablishMultipleLinksToIndexEntriesWithCleanupJob methodsFor: 'initialize-release'!
initialize

    super initialize.
    subscripts := OoVArray new.
    indexEntries := OoVArray new.
    tempIndexEntriesToDelete := OoVArray new.!
```

```
Smalltalk defineClass: #IndexEntryAddJob
	superclass: #{OneStepJob}
	indexedType: #none
	private: false
	instanceVariableNames: '
		indexEntry <IndexEntry>
		subscript <uint32>
		replyQuorumDenominator <uint32>
		replyTag <uint64>
		replyContentionIndex <uint16>
		actualNewData <ooTransient> '
	classInstanceVariableNames: ''
	imports: ''
	category: 'ENVY/Manager'!


!IndexEntryAddJob class methodsFor: 'private: generated'!
ooCodeGenVersion

	^ 1!
ooTypedInstanceVariablesString

	^ '
		indexEntry <IndexEntry>
		subscript <uint32>
		replyQuorumDenominator <uint32>
		replyTag <uint64>
		replyContentionIndex <uint16>
		actualNewData <ooTransient> '!
!


!IndexEntryAddJob methodsFor: 'accessing'!
actualNewData
	"Normally this gets transmitted via socket to avoid having to do a remote MROW read."

	actualNewData isNil ifTrue: [
		actualNewData := indexEntry copy].
	^actualNewData!
!


!IndexEntryAddJob methodsFor: 'checking'!
checkContentionSpace
	"Make sure I'll only access objects in my own contention space.  This check can be
	made at any time after the job has been fully initialized, including just prior to execution."

	[actualNewData contentionIndex = contentionIndex] assert.!
!


!IndexEntryAddJob methodsFor: 'execution'!
executeWith: aContentionSpaceProcess

	| newIndexEntry linkJob |
	newIndexEntry := self actualNewData.
	(newIndexEntry targetContainerIn: aContentionSpaceProcess session) ooCluster: newIndexEntry.
	linkJob := EstablishLinkToIndexEntryWithCleanupJob new
		indexedObject: newIndexEntry indexedObject;
		subscript: subscript;
		indexEntry: newIndexEntry;
		tempIndexEntryToDelete: indexEntry.
	linkJob
		tagInteger: replyTag;
		contentionIndex: replyContentionIndex;
		quorumFraction: 1 / replyQuorumDenominator.
	aContentionSpaceProcess addJob: linkJob.
"	observers do: [:each | each index: self hasAdded: indexEntry]"!
!
```

```
!IndexEntryAddJob methodsFor: 'accessing'!
indexEntry: anIndexEntry

	indexEntry := anIndexEntry.
	contentionIndex := anIndexEntry contentionIndex.
	actualNewData := anIndexEntry copy.!
!


!IndexEntryAddJob methodsFor: 'copying'!
postCopy

	super postCopy.
	self actualNewData.!
!


!IndexEntryAddJob methodsFor: 'passivation/activation'!
preSavePassivation
	"Answer an object to passivate in place of myself.  Subclasses should reimplement
	 if there is a need to translate the object in some way during passivation."

	self actualNewData.  "Make sure this transient instVar has a valid value prior to marshalling."
	^self!
!


!IndexEntryAddJob methodsFor: 'accessing'!
replyContentionIndex: anInteger

	replyContentionIndex := anInteger.!
replyQuorumDenominator: aDenominatorInteger

	replyQuorumDenominator := aDenominatorInteger.!
replyTag: replyTagInt

	replyTag := replyTagInt.!
subscript: sub

	subscript := sub.!
!

-----------------------------------------------------------------------------------------------------------------------

Smalltalk defineClass: #IndexEntryDeleteJob
	superclass: #{OneStepJob}
	indexedType: #none
	private: false
	instanceVariableNames: '
		indexedObject <Object>
		indexEntry <IndexEntry>
		replyTag <uint64>
		replyQuorumDenominator <uint32>
		objectContentionIndex <uint16>'
	classInstanceVariableNames: ''
	imports: ''
	category: ''!


!IndexEntryDeleteJob class methodsFor: 'private: generated'!
ooCodeGenVersion

	^ 1!
ooTypedInstanceVariablesString

	^'
		indexedObject <Object>
		indexEntry <IndexEntry>
		replyTag <uint64>
		replyQuorumDenominator <uint32>
		objectContentionIndex <uint16>'!
!
```

```
!IndexEntryDeleteJob methodsFor: 'checking'!
checkContentionSpace
	"Make sure I'll only access objects in my own contention space.  This check can be
	 made at any time after the job has been fully initialized, including just prior to execution."

	[indexEntry contentionIndex = contentionIndex] assert.!
!


!IndexEntryDeleteJob methodsFor: 'execution'!
executeWith: aContentionSpaceProcess
	"Delete the index entry and send back a IndexEntryHasBeenDeletedJob with a quorum fraction of
	 1 / numberOfIndexEntries.  This will synchronize with all the other DeleteUnindexedObjectJobs
	 sent back by my siblings, the other DeleteIndexEntryThenObjectJobs spawned by the original
	 DeleteObjectJob."

	| replyJob |
"	[indexEntry indexedObject == indexedObject] assert."
	indexEntry ooDelete.
	"Set up a reply job to indicate that the index entry was deleted."
	replyJob := IndexEntryHasBeenDeletedJob new
		indexedObject: indexedObject;
		contentionIndex: objectContentionIndex;
		tagInteger: replyTag;
		quorumFraction: 1 / replyQuorumDenominator.
	aContentionSpaceProcess addJob: replyJob.!
!


!IndexEntryDeleteJob methodsFor: 'accessing'!
indexedObject: theIndexedObject

	indexedObject := theIndexedObject.!
indexEntry: theIndexEntry

	indexEntry := theIndexEntry.!
objectContentionIndex: theObjectContentionIndex

	objectContentionIndex := theObjectContentionIndex.!
replyQuorumDenominator: quorum

	replyQuorumDenominator := quorum.!
replyTag: replyTagInt

	replyTag := replyTagInt.!
!

"-------------------------------------------------------------------------"
Smalltalk defineClass: #IndexEntryReplyJob
	superclass: #{OneStepJob}
	indexedType: #none
	private: false
	instanceVariableNames: '
		indexedObject <Object> '
	classInstanceVariableNames: ''
	imports: ''
	category: 'ENVY/Manager'!


!IndexEntryReplyJob class methodsFor: 'private: generated'!
ooCodeGenVersion

	^ 1!
ooTypedInstanceVariablesString

	^ '
		indexedObject <Object> '!
!
```

```
!IndexEntryReplyJob methodsFor: 'private: helpers'!
addSideEffectsTo: aMultipleLinkJob
	"Record any side-effects that I produce into a job designed specifically to update multiple pointers to
	 index entries and then to check for pending requests on the indexedObject."

	self subclassResponsibility.!
!


!IndexEntryReplyJob methodsFor: 'execution'!
collapseJobs: jobs
	"Given a collection of jobs which includes me, answer either nil or a new
	 single job that has the same effect as the given jobs.  Each job in the
	 collection will be given the chance to collapse, and the first job that replies
	 with a collapsed job will be the one that determines how to collapse them.
	 Since the jobs are persistent, this is always called within a transaction."

	"I expect to be grouped with other EstablishLinkToIndexEntryWithCleanupJobs.
	 Take the IndexEntries present in these jobs and plug them into the object's list
	 of known IndexEntries."

	| newJob |
	jobs assertAllType: IndexEntryReplyJob.

	newJob := EstablishMultipleLinksToIndexEntriesWithCleanupJob new.
	newJob indexedObject: indexedObject.
	newJob contentionIndex: contentionIndex.
	jobs do: [:j |
		j addSideEffectsTo: newJob].
	^newJob!
executeWith: aContentionSpaceProcess

	self error: 'This should have been collapsed'.  "(with others like me and/or EstablishLinkToIndexEntryWithCleanupJobs)"!
!


!IndexEntryReplyJob methodsFor: 'accessors'!
indexedObject: anObject

	indexedObject := anObject.!
!


!IndexEntryReplyJob methodsFor: 'execution'!
start

	self error: 'This job should have been collapsed.'.!
!

Smalltalk defineClass: #EstablishLinkToIndexEntryWithCleanupJob
	superclass: #{IndexEntryReplyJob}
	indexedType: #none
	private: false
	instanceVariableNames: '
		subscript <uint32>
		indexEntry <IndexEntry>
		tempIndexEntryToDelete <IndexEntry> '
	classInstanceVariableNames: ''
	imports: ''
	category: 'ENVY/Manager'!


!EstablishLinkToIndexEntryWithCleanupJob class methodsFor: 'private: generated'!
ooCodeGenVersion

	^ 1!
ooTypedInstanceVariablesString

	^ '
		subscript <uint32>
		indexEntry <IndexEntry>
		tempIndexEntryToDelete <IndexEntry> '!
!
```

```
!EstablishLinkToIndexEntryWithCleanupJob methodsFor: 'private: helpers'!
addSideEffectsTo: aMultipleLinkJob
	"Record any side-effects that I produce into a job designed specifically to update multiple pointers to
	 index entries and then to check for pending requests on the indexedObject."

	aMultipleLinkJob assertType: EstablishMultipleLinksToIndexEntriesWithCleanupJob.
	aMultipleLinkJob
		addSubscript: subscript
		newIndexEntry: indexEntry.
	aMultipleLinkJob
		addIndexEntryToDelete: tempIndexEntryToDelete.!
!


!EstablishLinkToIndexEntryWithCleanupJob methodsFor: 'accessors'!
indexEntry: anIndexEntry

	indexEntry := anIndexEntry.!
subscript: sub

	subscript := sub.!
tempIndexEntryToDelete: tempIndexEntry

	tempIndexEntryToDelete := tempIndexEntry.!
!

Smalltalk defineClass: #IndexEntryHasBeenDeletedJob
	superclass: #{IndexEntryReplyJob}
	indexedType: #none
	private: false
	instanceVariableNames: "
	classInstanceVariableNames: "
	imports: "
	category: "!


!IndexEntryHasBeenDeletedJob methodsFor: 'private: helpers'!
addSideEffectsTo: aMultipleLinkJob
	"Record any side-effects that I produce into a job designed specifically to update multiple pointers to
	 index entries and then to check for pending requests on the indexedObject."

	aMultipleLinkJob assertType: EstablishMultipleLinksToIndexEntriesWithCleanupJob.

	"Do nothing.  This job simply acts as synchronization."!
!

Smalltalk defineClass: #IndexEntryHasBeenUpdatedJob
	superclass: #{IndexEntryReplyJob}
	indexedType: #none
	private: false
	instanceVariableNames: '
		tempIndexEntryToDelete <IndexEntry>'
	classInstanceVariableNames: "
	imports: "
	category: "!


!IndexEntryHasBeenUpdatedJob class methodsFor: 'private: generated'!
ooCodeGenVersion

	^ 1!
ooTypedInstanceVariablesString

	^'
		tempIndexEntryToDelete <IndexEntry>'!
!
```

```
!IndexEntryHasBeenUpdatedJob methodsFor: 'private: helpers'!
addSideEffectsTo: aMultipleLinkJob
      "Record any side-effects that I produce into a job designed specifically to update multiple pointers to
       index entries and then to check for pending requests on the indexedObject."

      aMultipleLinkJob assertType: EstablishMultipleLinksToIndexEntriesWithCleanupJob.

      aMultipleLinkJob addIndexEntryToDelete: tempIndexEntryToDelete.!
!


!IndexEntryHasBeenUpdatedJob methodsFor: 'accessors'!
tempIndexEntryToDelete: anIndexEntry

      tempIndexEntryToDelete := anIndexEntry.!
!

"-------------------------------------------------------------------------"
Smalltalk defineClass: #IndexEntryUpdateJob
      superclass: #{OneStepJob}
      indexedType: #none
      private: false
      instanceVariableNames: '
            indexEntryToUpdate <IndexEntry>
            newIndexEntryData <IndexEntry>
            replyTag <uint64>
            replyQuorumDenominator <uint32>
            replyContentionIndex <uint16>
            actualNewData <ooTransient> '
      classInstanceVariableNames: ''
      imports: ''
      category: 'ENVY/Manager'!


!IndexEntryUpdateJob class methodsFor: 'private: generated'!
ooCodeGenVersion

      ^ 1!
ooTypedInstanceVariablesString

      ^'
            indexEntryToUpdate <IndexEntry>
            newIndexEntryData <IndexEntry>
            replyTag <uint64>
            replyQuorumDenominator <uint32>
            replyContentionIndex <uint16>
            actualNewData <ooTransient> '!
!


!IndexEntryUpdateJob methodsFor: 'accessing'!
actualNewData
      "Normally this gets transmitted via socket to avoid having to do a remote MROW read."

      actualNewData isNil ifTrue: [
            actualNewData := newIndexEntryData copy].
      ^actualNewData!
!


!IndexEntryUpdateJob methodsFor: 'checking'!
checkContentionSpace
      "Make sure I'll only access objects in my own contention space.  This check can be
       made at any time after the job has been fully initialized, including just prior to execution."

      [((OoDB of: indexEntryToUpdate ooContainer) systemName readStream upTo: $-; upToEnd) trimBlanks asInteger= contentionIndex] assert.
      [actualNewData contentionIndex = contentionIndex] assert.!
!
```

```
!IndexEntryUpdateJob methodsFor: 'execution'!
executeWith: aContentionSpaceProcess
	"Update the index entry, then send a job back to the original object (at replyContentionIndex) to
	 delete the temporary index entry that we used as the source of our data for this update."

	| replyJob |
	indexEntryToUpdate assignFrom: self actualNewData.
	replyJob := IndexEntryHasBeenUpdatedJob new
		contentionIndex: replyContentionIndex;
		indexedObject: indexEntryToUpdate indexedObject;
		tempIndexEntryToDelete: newIndexEntryData;
		tagInteger: replyTag;
		quorumFraction: 1 / replyQuorumDenominator.
	aContentionSpaceProcess addJob: replyJob.!
!


!IndexEntryUpdateJob methodsFor: 'accessing'!
indexEntryToUpdate: anIndexEntry

	indexEntryToUpdate := anIndexEntry.!
newIndexEntryData: anIndexEntry

	newIndexEntryData := anIndexEntry.
	actualNewData := anIndexEntry copy.!
!


!IndexEntryUpdateJob methodsFor: 'copying'!
postCopy

	super postCopy.
	self actualNewData.!
!


!IndexEntryUpdateJob methodsFor: 'passivation/activation'!
preSavePassivation
	"Answer an object to passivate in place of myself.  Subclasses should reimplement
	 if there is a need to translate the object in some way during passivation."

	self actualNewData.  "Make sure this transient instVar has a valid value prior to marshalling."
	^self!
!


!IndexEntryUpdateJob methodsFor: 'accessing'!
replyContentionIndex: anInteger

	replyContentionIndex := anInteger.!
replyQuorumDenominator: anInteger

	replyQuorumDenominator := anInteger.!
replyTag: anInteger

	replyTag := anInteger.!
!


Smalltalk defineClass: #RandomChangeForSaturationTestJob
	superclass: #{OneStepJob}
	indexedType: #none
	private: false
	instanceVariableNames: '
		productSubscript <uint32> '
	classInstanceVariableNames: ''
	imports: ''
	category: ''!
```

```
!RandomChangeForSaturationTestJob class methodsFor: 'private: generated'!
ooCodeGenVersion

    ^ 1!
ooTypedInstanceVariablesString

    ^ '
        productSubscript <uint32> '!
!


!RandomChangeForSaturationTestJob methodsFor: 'execution'!
executeWith: aContentionSpaceProcess
    "Make a random change to the product object I specify, causing the object to be reindexed if necessary."

    | rootObject |
    rootObject := CreateFakeObjectsJob pickProductUsing: productSubscript contentionSpaceProcess: aContentionSpaceProcess.
    rootObject isStable
        ifTrue: [
            | oldTransients |
            oldTransients := rootObject transientIndexEntries.
            self makeRandomChangeTo: rootObject.
            IndexEntryTracker new
                contentionSpaceProcess: aContentionSpaceProcess
                object: rootObject
                objectContentionIndex: contentionIndex
                oldTransientsIfKnown: oldTransients]
        ifFalse: [
            rootObject isDeleting ifTrue: [^self].  "Ignore changes to doomed objects."
            [rootObject isReindexing] assert. "not stable means either deleting or reindexing."
            "The reindexing will happen after the current reindexing completes."
            self makeRandomChangeTo: rootObject.
            rootObject requestReindexing].!
!


!RandomChangeForSaturationTestJob methodsFor: 'private'!
makeRandomChangeTo: aPerson
    "Make a random change to the product object I specify.  Don't trigger reindexing from here.
    The change should have the following distribution:
        1           new call record
        1/15,000    change address
        1/150,000   change name
        1/2,000     change plan
        1/10,000    change line
        1/300       make payment.
    To accomodate those relative frequencies, the relative proportions are mapped onto ranges
    of values in the range 0..1.  The total of these ratio values is:
        1.0d+(1/15000)+(1/150000)+(1/2000)+(1/10000)+(1/300) = 1.0040066666667d
    Whoops - running short on time now.  I'll simplify the test and make *every* change be
    a call record."

    | callRecord |
    callRecord := Telecom_OutboundCallRecord new.  "make all records outbound to simplify"
    callRecord
        start: Timestamp now;
        duration: 120;
        calledLine: 2125551212.
    aPerson accounts first lines first addCall: callRecord.!
!


!RandomChangeForSaturationTestJob methodsFor: 'accessing'!
productSubscript: anInteger
    "Set my subscript.  This simply indexes into the contention space's RAM cache of all products.
    This whole class is only to be used for computing saturation throughput of the USP."

    productSubscript := anInteger.!
!
```

```
Smalltalk defineClass: #RecacheProductsJob
	superclass: #{OneStepJob}
	indexedType: #none
	private: false
	instanceVariableNames: ''
	classInstanceVariableNames: ''
	imports: ''
	category: ''!


!RecacheProductsJob methodsFor: 'execution'!
executeWith: aContentionSpaceProcess
	"Regenerate the cached collection of all persistent products in this contention space."

	CreateFakeObjectsJob recacheRootObjectsFor: aContentionSpaceProcess.!
!

Smalltalk defineClass: #ComputedAttribute
	superclass: #{SchemaAttribute}
	indexedType: #none
	private: false
	instanceVariableNames: '
		computationPolicy <ooShortRef(ComputationPolicy)>
		formula <Formula>'
	classInstanceVariableNames: ''
	imports: ''
	category: 'ENVY/Manager'!


!ComputedAttribute class methodsFor: 'loading'!
decodeAsLiteralArray: anArray withSchema: schema
	"Return an instance based on the information encoded in anArray and the given schema."

	| attribute |
	attribute := super
		decodeAsLiteralArray: (anArray copyFrom: 1 to: anArray size - 2)
		withSchema: schema.
	attribute formula: (anArray at: anArray size - 1) decodeAsLiteralArray.
	attribute computationPolicy: (anArray at: anArray size) decodeAsLiteralArray.
	^attribute!
!


!ComputedAttribute class methodsFor: 'private: generated'!
ooCodeGenVersion

	^ 1!
ooTypedInstanceVariablesString

	^'
		computationPolicy <ooShortRef(ComputationPolicy)>
		formula <Formula>'!
!


!ComputedAttribute methodsFor: 'accessing'!
computationPolicy

	^computationPolicy!
computationPolicy: aComputationPolicy

	aComputationPolicy assertType: ComputationPolicy.
	computationPolicy := aComputationPolicy.!
formula

	^formula!
formula: aFormula

	aFormula assertType: Formula.
	formula := aFormula.!
!
```

```
!ComputedAttribute methodsFor: 'testing'!
isComputed

    ^true!
!


!ComputedAttribute methodsFor: 'dumping'!
literalArrayEncoding
    "Convert me to a literal array."

    ^super literalArrayEncoding,
        (Array
            with: self formula literalArrayEncoding
            with: self computationPolicy literalArrayEncoding)!
!
```

```
Smalltalk defineClass: #UncomputedAttribute
    superclass: #{SchemaAttribute}
    indexedType: #none
    private: false
    instanceVariableNames: ''
    classInstanceVariableNames: ''
    imports: ''
    category: 'ENVY/Manager'!


!UncomputedAttribute methodsFor: 'testing'!
isComputed

    ^false!
!
```

```
Smalltalk defineClass: #IndexedSchemaClass
    superclass: #{SchemaClass}
    indexedType: #none
    private: false
    instanceVariableNames: '
        indices <OoVArray(Index)>'
    classInstanceVariableNames: ''
    imports: ''
    category: 'ENVY/Manager'!


!IndexedSchemaClass class methodsFor: 'private: generated'!
ooCodeGenVersion

    ^ 1!
ooTypedInstanceVariablesString

    ^'
        indices <OoVArray(Index)>'!
!
```

```
!IndexedSchemaClass methodsFor: 'accessing'!
allUniqueIndices
	"Answer a collection containing indices with unique keys for myself and my superclasses
	 and my subclasses."

	| uniques |
	uniques := OrderedCollection new.
	superClass notNil ifTrue: [
		uniques addAll: superClass applicableUniqueIndices].
	uniques addAll: self uniqueIndices.
	self allSubclasses do: [:sub |
		uniques addAll: sub uniqueIndices].
	^uniques asArray!
applicableUniqueIndices
	"Answer a collection containing indices for myself and my superclasses that must have unique keys.
	 Don't look at my subclasses."

	| myUnique |
	myUnique := self uniqueIndices.
	superClass notNil
		ifTrue: [^superClass applicableUniqueIndices, myUnique]
		ifFalse: [^myUnique].!
indices

	^indices!
indices: someIndices

	someIndices do: [:ind | ind assertType: Index].
	indices == someIndices ifTrue: [^self].
	indices replaceWithElements: someIndices.
	self ooUpdate. "Compensate for Objectivity bug with empty OoVArrays."!
!


!IndexedSchemaClass methodsFor: 'initialize-release'!
initialize

	super initialize.
	indices := OoVArray new.
	^self!
!


!IndexedSchemaClass methodsFor: 'maintenance'!
replaceAttribute: oldAttribute with: newAttribute
	"Replace the old attribute with a new one.  For every reference from the schema's subobjects to
	 the old attribute, update it to refer to the new attribute."

	super replaceAttribute: oldAttribute with: newAttribute.
	indices do: [:role | role replaceAttribute: oldAttribute with: newAttribute].!
!


!IndexedSchemaClass methodsFor: 'accessing'!
uniqueIndices
	"Answer a collection containing just my indices that must have unique keys.  Don't look at
	 my superclasses and subclasses."

	^indices select: [:ind | ind isUnique]!
!


Smalltalk defineClass: #RoleObject
	superclass: #{Core.Object}
	indexedType: #none
	private: false
	instanceVariableNames: '
		localObject <ooShortRef(Object)> '
	classInstanceVariableNames: 'cachedIndex '
	imports: ''
	category: 'ENVY/Manager'!
```

```
!RoleObject class methodsFor: 'private: generated'!
ooCodeGenVersion

    ^ 1!
ooTypedInstanceVariablesString

    ^'
        localObject <ooShortRef(Object)> '!
!


!RoleObject methodsFor: 'accessing'!
attachTo: anotherRoleObject
    "Tell the other role object to plug me in as its neighbour."

    anotherRoleObject assertType: RoleObject.
    self subclassResponsibility.!
connectedRoles
    "Answer a collection of currently connected neighbouring roles (exclude myself, of course).
     Include nils as placeholders for roles that have not been connected yet."

    self subclassResponsibility.!
isFullyConnectedToOtherRoles

    ^(self connectedRoles includes: nil) not!
localObject

    ^localObject!
localObject: anObject

    localObject := anObject.!
!


Smalltalk defineClass: #SchemaLookupStructure
    superclass: #{Core.Object}
    indexedType: #none
    private: false
    instanceVariableNames: '
        dataPaths <ooVArray(ooShortRef(AttributePath))>
        keys <ooVArray(ooShortRef(AttributePath))> '
    classInstanceVariableNames: ''
    imports: ''
    category: 'ENVY/Manager'!


!SchemaLookupStructure class methodsFor: 'instance creation'!
keys: keyAttributePaths dataPaths: dataAttributePaths

    ^self new
        keys: keyAttributePaths;
        dataPaths: dataAttributePaths;
        yourself!
new

    ^super new initialize!
!


!SchemaLookupStructure class methodsFor: 'private: generated'!
ooCodeGenVersion

    ^ 1!
ooTypedInstanceVariablesString

    ^'
        dataPaths <ooVArray(ooShortRef(AttributePath))>
        keys <ooVArray(ooShortRef(AttributePath))> '!
!
```

```
!SchemaLookupStructure methodsFor: 'accessing'!
addDataPath: anAttributePath

    anAttributePath assertType: AttributePath.
    dataPaths addElement: anAttributePath.
    self ooUpdate. "Compensate for Objectivity bug with empty OoVArrays."
    self changed: #dataPaths.!
addKey: anAttributePath

    anAttributePath assertType: AttributePath.
    keys addElement: anAttributePath..
    self ooUpdate. "Compensate for Objectivity bug with empty OoVArrays."
    self changed: #keys.!
attributePathIsLegalKey: anAttributePath
    "Answer whether the given attribute path would make a legal key."

    anAttributePath attributes do: [:attr |
        attr cardinality isMany ifTrue: [^false]].
    ^anAttributePath attributes last type isPrimitive!
!


!SchemaLookupStructure methodsFor: 'maintenance'!
cleanUpIllegalPathsWithType: aSchemaClass

    self dataPaths: (self dataPaths select: [:path |
        path isLegalPathForIndex: 1 type: aSchemaClass]).
    self keys: (self keys select: [:path |
        path isLegalPathForIndex: 1 type: aSchemaClass]).!
cleanupRemovedAttribute: anAttribute
    "This SchemaAttribute has been removed.  Clean up my indices and roles."

    self dataPaths:
        (self dataPaths reject: [:path |
            path attributes includes: anAttribute]).
    self keys:
        (self keys reject: [:path |
            path attributes includes: anAttribute]).!
!


!SchemaLookupStructure methodsFor: 'accessing'!
dataPaths

    ^dataPaths!
dataPaths: someAttributePaths

    someAttributePaths assertAllType: AttributePath.
    dataPaths replaceWithElements: someAttributePaths.
    self ooUpdate. "Compensate for Objectivity bug with empty OoVArrays."
    self changed: #dataPaths.!
!


!SchemaLookupStructure methodsFor: 'initialize-release'!
initialize

    keys := OoVArray new.
    dataPaths := OoVArray new.!
!


!SchemaLookupStructure methodsFor: 'accessing'!
isUnique

    ^false!
keys

    ^keys!
```

```
keys: someAttributePaths

        someAttributePaths assertAllType: AttributePath.
        keys replaceWithElements: someAttributePaths.
        self ooUpdate. "Compensate for Objectivity bug with empty OoVArrays."
        self changed: #keys.!
numberOfContainers

        ^0!
!


!SchemaLookupStructure methodsFor: 'copying'!
postCopy

        | mapping |
        super postCopy.
        mapping := IdentityDictionary new.
        dataPaths := dataPaths collect: [:path |
                mapping at: path ifAbsentPut: [path copy]].
        keys := keys collect: [:path |
                mapping at: path ifAbsentPut: [path copy]].!
!


!SchemaLookupStructure methodsFor: 'accessing'!
removeDataPath: anAttributePath
        "Note that equality is used to locate the actual AttributePath to remove."

        anAttributePath assertType: AttributePath.
        dataPaths removeElementByEquality: anAttributePath.
        self ooUpdate. "Compensate for Objectivity bug with empty OoVArrays."
        self changed: #dataPaths.!
removeKey: anAttributePath
        "Note that equality is used to locate the actual AttributePath to remove."

        anAttributePath assertType: AttributePath.
        keys removeElementByEquality: anAttributePath.
        self ooUpdate. "Compensate for Objectivity bug with empty OoVArrays."
        self changed: #keys.!
!


!SchemaLookupStructure methodsFor: 'maintenance'!
replaceAttribute: oldAttribute with: newAttribute
        "Replace the old attribute with a new one.  For every reference from the schema's subobjects to
         the old attribute, update it to refer to the new attribute."

        dataPaths do: [:path |
                path replaceAttribute: oldAttribute with: newAttribute].
        keys do: [:path |
                path replaceAttribute: oldAttribute with: newAttribute].!
!


Smalltalk defineClass: #Index
        superclass: #{SchemaLookupStructure}
        indexedType: #none
        private: false
        instanceVariableNames: '
                name <OoVString>
                isUnique <boolean>
                observers <OoVArray(IndexObserver)> '
        classInstanceVariableNames: ''
        imports: ''
        category: 'ENVY/Manager'!
```

```
!Index class methodsFor: 'loading'!
decodeAsLiteralArray: anArray withSchema: schema forClass: cls
	"Return an instance based on the information encoded in anArray and the
	 given Schema and SchemaClass."

	| nameString indexObservers dataPaths keyPaths instance isUnique |
	[anArray size = 6] assert.
	nameString := (anArray at: 2).
	isUnique := (anArray at: 3).
	indexObservers := (anArray at: 4) collect: [:arr | arr decodeAsLiteralArray].
	dataPaths := (anArray at: 5) collect: [:arr |
		| attrs currentType |
		currentType := cls.
		attrs := OrderedCollection new: arr size.
		arr do: [:attrName |
			attrs addLast: (currentType allAttributes detect: [:x | x name = attrName]).
			currentType := attrs last type].
		AttributePath new attributes: attrs].
	keyPaths := (anArray at: 6) collect: [:arr |
		| attrs currentType |
		currentType := cls.
		attrs := OrderedCollection new: arr size.
		arr do: [:attrName |
			attrs addLast: (currentType allAttributes detect: [:x | x name = attrName]).
			currentType := attrs last type].
		AttributePath new attributes: attrs].
	instance := self
		name: nameString
		observers: indexObservers
		keys: keyPaths
		dataPaths: dataPaths.
	instance isUnique: isUnique.
	^instance!
!


!Index class methodsFor: 'instance creation'!
fromIndex: anIndex
	"Answer a new instance of me that looks like anIndex.  Feel free to reuse pieces of
	 anIndex, as it will not be used after this call."

	^self
		name: anIndex name
		observers: anIndex observers
		keys: anIndex keys
		dataPaths: anIndex dataPaths!
!


!Index class methodsFor: 'class accessing'!
mechanisms
	"This should be rewritten some day to be more pluggable."

	^Dictionary new
		at: 'One Container' put: ContainerIndex;
		at: 'Parallel Index' put: ParallelIndex;
		yourself!
!

!Index class methodsFor: 'instance creation'!
name: aString observers: indexObservers keys: keyAttributePaths dataPaths: dataAttributePaths

	^(self keys: keyAttributePaths dataPaths: dataAttributePaths)
		name: aString;
		observers: indexObservers;
		yourself!
!
```

```
!Index class methodsFor: 'private: generated'!
ooCodeGenVersion

    ^ 1!
ooTypedInstanceVariablesString

    ^ '
        name <OoVString>
        isUnique <boolean>
        observers <OoVArray(IndexObserver)> '!
!


!Index methodsFor: 'accessing'!
addObserver: anObserver

    anObserver assertType: IndexObserver.
    observers addElement: anObserver.
    self ooUpdate. "Compensate for Objectivity bug with empty OoVArrays."
    self changed: #observers.!
at: subscript putContainer: aContainer contentionIndex: contentionIndex

    self subclassResponsibility.!
!


!Index methodsFor: 'accessing-ui'!
attributePathIsLegalContentionKey: anAttributePath
    "Answer whether the given attribute path would make a legal contention key."

    anAttributePath attributes do: [:attr |
        attr cardinality isMany ifTrue: [^false].
        attr type hasIdentity ifTrue: [^false]].
    ^anAttributePath attributes last type isPrimitive!
attributePathIsLegalKey: anAttributePath
    "Answer whether the given attribute path would make a legal key."

    "Although plain lookup structures can't use attributes with cardinality many in the
     chain of attributes for a key, an Index can.  That's because many IndexEntries
     may be created to index one object, whereas the lookup structures might not be
     able to fan out that way."

    ^anAttributePath attributes last type isPrimitive!
!


!Index methodsFor: 'accessing'!
containerFor: anIndexEntry inSession: session
    "Answer the OoContainer in which one would find anIndexEntry."

    self subclassResponsibility.!
contentionSpaceFor: anIndexEntry
    "Answer which contention space should contain anIndexEntry."

    self subclassResponsibility.!
!


!Index methodsFor: 'printing'!
displayString

    ^name!
!


!Index methodsFor: 'initialize-release'!
initialize

    super initialize.
    name := String new.
    observers := OoVArray new.
    isUnique := true.!
!
```

```
!Index methodsFor: 'accessing'!
isUnique

	^isUnique!
isUnique: aBoolean

	isUnique := aBoolean.!
!


!Index methodsFor: 'dumping'!
literalArrayEncoding
	"Convert me to a literal array."

	^(Array new: 6)
		at: 1 put: self class fullyQualifiedReference;
		at: 2 put: self name;
		at: 3 put: self isUnique;
		at: 4 put: (observers asArray collect: [:ob | ob literalArrayEncoding]);
		at: 5 put: (self dataPaths asArray collect: [:path |
			path attributes collect: [:attr | attr name]]);
		at: 6 put: (self keys asArray collect: [:path |
			path attributes collect: [:attr | attr name]]);
		yourself!
!


!Index methodsFor: 'accessing'!
name

	^name!
name: aString

	name := aString.!
observers

	^observers!
observers: someObservers

	someObservers assertAllType: IndexObserver.
	observers replaceWithElements: someObservers.
	self ooUpdate. "Compensate for Objectivity bug with empty OoVArrays."
	self changed: #observers.!
!


!Index methodsFor: 'copying'!
postCopy

	super postCopy.
	observers := observers copy.!
!
```

```
!Index methodsFor: 'printing'!
printShortDescriptionOn: aTextStream internal: isInternal stereotypes: stereotypes
    "Write a short description of me to the text stream.  Make it suitable for presentation in a user interface."

    | order |
    name isEmpty ifFalse: [
        aTextStream emphasis: #italic; nextPutAll: name; emphasis: nil.
        stereotypes notEmpty ifTrue: [
            aTextStream space; nextPut: (Character value: 171).  "open chevron"
            stereotypes
                do: [:str | aTextStream nextPutAll: str]
                separatedBy: [aTextStream nextPutAll: ', '].
            aTextStream nextPut: (Character value: 187)]].  "close chevron"
    [self keys all: [:key | self dataPaths includes: key]] assert: 'Inconsistent index definition'.
    order := self keys asSortedCollection asArray,
        (self dataPaths asSet - self keys asSet) asSortedCollection asArray.
    order
        do: [:sourcePath |
            aTextStream cr.
            isInternal ifTrue: [aTextStream tab].
            (self keys includes: sourcePath)
                ifTrue: [aTextStream emphasis: #bold]
                ifFalse: [aTextStream emphasis: nil].
            sourcePath attributes
                do: [:subAttr | aTextStream nextPutAll: subAttr name]
                separatedBy: [aTextStream nextPutAll: '-']]
        separatedBy: [aTextStream nextPutAll: ','].
    aTextStream emphasis: nil.!
!


!Index methodsFor: 'accessing'!
removeObserver: anObserver

    anObserver assertType: IndexObserver.
    observers removeElement: anObserver.
    self ooUpdate.  "Compensate for Objectivity bug with empty OoVArrays."
    self changed: #observers.!
!


!Index methodsFor: 'initialize-release'!
setName: aString observers: indexObservers keys: keyAttributePaths instanceVariableSources: ivarAttributePaths

    name := aString.
    observers := indexObservers asOoVArray.
    keys := keyAttributePaths asOoVArray.
    dataPaths := ivarAttributePaths asOoVArray.
    ^self!
!


Smalltalk defineClass: #ContainerIndex
    superclass: #{Index}
    indexedType: #none
    private: false
    instanceVariableNames: '
        contentionIndex <uint32>
        targetContainerNumber <uint32>'
    classInstanceVariableNames: ''
    imports: ''
    category: 'ENVY/Manager'!


!ContainerIndex class methodsFor: 'private: generated'!
ooCodeGenVersion

    ^1!
ooTypedInstanceVariablesString

    ^'
        contentionIndex <uint32>
        targetContainerNumber <uint32>'!
!
```

```
!ContainerIndex methodsFor: 'accessing'!
at: subscript putContainerNumber: aContainerNumber contentionIndex: newContentionIndex

    [subscript = 1] assert.

    [targetContainerNumber = 0] assert.
    targetContainerNumber := aContainerNumber.

    [contentionIndex = 0] assert.  "uninitialized"
    [newContentionIndex > 0] assert.  "1..max"
    contentionIndex := newContentionIndex.!
containerFor: anIndexEntry inSession: session
    "Answer the OoContainer in which one would find anIndexEntry.  Called in a transaction of session."

    anIndexEntry assertType: IndexEntry.
    ^session ooProvideContainerFor: targetContainerNumber with: 'ignored'!
contentionSpaceFor: anIndexEntry
    "Answer which contention space should contain anIndexEntry."

    ^contentionIndex!
!


!ContainerIndex methodsFor: 'private: instance creation'!
initialize

    super initialize.
    contentionIndex := 0.  "invalid, indicating containers have not been assigned yet."!
!


!ContainerIndex methodsFor: 'accessing'!
numberOfContainers

    ^1!
!

Smalltalk defineClass: #ParallelIndex
    superclass: #{Index}
    indexedType: #none
    private: false
    instanceVariableNames: '
        contentionIndexes <ooVArray(uint32)>
        numberOfContainers <uint32>
        targetContainerNumbers <ooVArray(uint32)> '
    classInstanceVariableNames: ''
    imports: ''
    category: 'ENVY/Manager'!


!ParallelIndex class methodsFor: 'loading'!
decodeAsLiteralArray: anArray withSchema: schema forClass: cls
    "Return an instance based on the information encoded in anArray and the
     given Schema and SchemaClass."

    | index |
    index := super
        decodeAsLiteralArray: (anArray copyFrom: 1 to: anArray size - 1)
        withSchema: schema
        forClass: cls.
    index numberOfContainers: anArray last.
    ^index!
!
```

```
!ParallelIndex class methodsFor: 'instance creation'!
defaultNumberOfContainers

    ^5!
fromIndex: anIndex
    "Answer a new instance of me that looks like anIndex.  Feel free to reuse pieces of
     anIndex, as it will not be used after this call."

    (anIndex isKindOf: ParallelIndex) ifTrue: [^anIndex].
    ^self
        name: anIndex name
        observers: anIndex observers
        keys: anIndex keys
        dataPaths: anIndex dataPaths!
name: aString observers: indexObservers keys: keyAttributePaths dataPaths: dataAttributePaths numberOfContainers: anInteger

    ^(super
            name: aString
            observers: indexObservers
            keys: keyAttributePaths
            dataPaths: dataAttributePaths)
        numberOfContainers: anInteger;
        yourself!
!


!ParallelIndex class methodsFor: 'private: generated'!
ooCodeGenVersion

    ^1!
ooTypedInstanceVariablesString

    ^'
        contentionIndexes <ooVArray(uint32)>
        numberOfContainers <uint32>
        targetContainerNumbers <ooVArray(uint32)> '!
!


!ParallelIndex methodsFor: 'accessing'!
at: subscript putContainerNumber: aContainerNumber contentionIndex: contentionIndex

    [subscript between: 1 and: numberOfContainers] assert.
    targetContainerNumbers size = 0 ifTrue: [
        [contentionIndexes size = 0] assert.
        targetContainerNumbers replaceWithElements: (Array new: numberOfContainers withAll: 0).
        contentionIndexes replaceWithElements: (Array new: numberOfContainers withAll: 0).
        self ooUpdate].

    [(targetContainerNumbers at: subscript) = 0] assert.
    targetContainerNumbers at: subscript put: aContainerNumber.

    [(contentionIndexes at: subscript) = 0] assert.
    [contentionIndex > 0] assert.  "1..max"
    contentionIndexes at: subscript put: contentionIndex!
containerFor: anIndexEntry inSession: session
    "Answer the OoContainer in which one would find anIndexEntry.  Called in a transaction of session."

    | containerNum |
    anIndexEntry assertType: IndexEntry.
    containerNum := targetContainerNumbers at: anIndexEntry hashOfKey - 1 \\ numberOfContainers + 1.
    ^session
        ooProvideContainerFor: containerNum
        with: 'ignored'!
containersHaveBeenCreated

    ^targetContainerNumbers size = numberOfContainers!
contentionSpaceFor: anIndexEntry
    "Answer which contention space should contain anIndexEntry."

    anIndexEntry assertType: IndexEntry.
    ^contentionIndexes at: anIndexEntry hashOfKey - 1 \\ numberOfContainers + 1!
!
```

```
!ParallelIndex methodsFor: 'initialize-release'!
initialize

	super initialize.
	targetContainerNumbers := OoVArray new.
	contentionIndexes := OoVArray new.
	numberOfContainers := self class defaultNumberOfContainers.!
!


!ParallelIndex methodsFor: 'dumping'!
literalArrayEncoding
	"Convert me to a literal array."

	^super literalArrayEncoding copyWith: numberOfContainers!
!


!ParallelIndex methodsFor: 'accessing'!
numberOfContainers

	^numberOfContainers!
numberOfContainers: anInteger

	[targetContainerNumbers isEmpty] assert: 'Can"t change index dimensions after creation'.
	numberOfContainers := anInteger.!
!


!ParallelIndex methodsFor: 'copying'!
postCopy

	super postCopy.
	targetContainerNumbers := targetContainerNumbers copy.
	contentionIndexes := contentionIndexes copy.!
!


Smalltalk.Applications defineClass: #IndexingFramework
	superclass: #{ENVY.Application}
	indexedType: #none
	private: false
	instanceVariableNames: ''
	classInstanceVariableNames: ''
	imports: ''
	category: 'ENVY/Manager'!


!Core.Array methodsFor: 'printing'!
printCompletelyOn: aStream indent: indent
	"Print all of my elements, not just the first few thousand bytes, onto aStream.  Indent
	 successive subarrays by the given amount as appropriate."

	self size < 20 ifTrue: [
		| temp multiline |
		temp := WriteStream on: (String new: 100).
		temp nextPut: $(.
		multiline := (1 to: self size) any: [:i |
			(self at: i) printCompletelyOn: temp indent: indent.
			i = self size ifFalse: [temp space].
			temp position > 100].
		multiline ifFalse: [
			temp nextPut: $).
			aStream nextPutAll: temp contents.
			^self]].
	aStream nextPut: $(.
	1 to: self size do: [:i |
		aStream crtab: indent.
		(self at: i) printCompletelyOn: aStream indent: indent + 1].
	aStream nextPut: $).!
!
```

```
!Core.Object methodsFor: 'printing'!
printCompletelyOn: aStream indent: indent
	"Print me, not just the first few thousand bytes, onto aStream.  Indent
	 successive subarrays by the given amount as appropriate."

	self printOn: aStream.  "Note - arrays embedded within other collections will not work."!
!
```

```
!Schema class methodsFor: 'loading'!
decodeFromLiteralArray: anArray
	"Return an instance based on the information encoded in anArray."

	| schema |
	[anArray size = 8] assert.

	schema := self new.
	schema lastSave: (Timestamp fromMilliseconds: (anArray at: 2) decodeAsLiteralArray).

	"Create the bare classes..."
	(anArray at: 3) do: [:array |
		| cls |
		cls := array decodeAsLiteralArray.
		schema addSchemaClass: cls named: cls name].

	"Link up inheritance..."
	(anArray at: 4) do: [:pair |
		| childClass parentClass |
		childClass := pair first.
		parentClass := pair last.
		parentClass notNil ifTrue: [
			(schema schemaClassNamed: childClass)
				superClass: (schema schemaClassNamed: parentClass)]].

	"Link up the attributes..."
	(anArray at: 5) do: [:pair |
		| sourceClass array attr |
		sourceClass := pair first.
		array := pair last.
		attr := array first asQualifiedReference value
			decodeAsLiteralArray: array
			withSchema: schema.
		(schema schemaClassNamed: sourceClass) addAttribute: attr].

	"Set up the class indices..."
	(anArray at: 6) do: [:triple |
		| cls kind index |
		cls := schema schemaClassNamed: triple first.
		kind := triple at: 2.
		index := triple last.
		index := index first asQualifiedReference value
			decodeAsLiteralArray: index
			withSchema: schema
			forClass: cls.
		kind = #recordIndex ifTrue: [
			self error: 'Record indices are no longer supported'].
		[kind = #objectIndex] assert.
		cls indices: (cls indices copyWith: index)].

	"Set up the roots..."
	(anArray at: 7) do: [:rootTypeName |
		| rootAttrName attr |
		rootAttrName := rootTypeName asString copy.
		rootAttrName at: 1 put: rootAttrName first asLowercase.
		attr := SchemaAttribute
			name: rootAttrName asSymbol
			type: (schema schemaClassNamed: rootTypeName).
		attr single.
		schema addRoot: attr].

	"Set up the relationships..."
	(anArray at: 8) do: [:subarray |
		| relationship |
		[subarray size = 4] assert.
		relationship := SchemaRelationship
			decodeAsLiteralArray: subarray
			withSchema: schema.
		schema addRelationship: relationship].

	^schema!
!
```

```
!Schema methodsFor: 'dumping'!
dumpString
	"Convert me into a dumped representation which, when compiled as Smalltalk code,
	 produces a Schema like the receiver."

	| str encoding |
	encoding := self literalArrayEncoding.
	[encoding size = 8] assert.

	str := WriteStream on: (String new: 1000).
	str nextPutAll: '"Schema definition file"'; cr; cr.
	str nextPutAll: '#('; crtab; print: (encoding at: 1).

	str crtab; nextPutAll: '"Last saved:" '.
	str print: (encoding at: 2); nextPutAll: ' "= ', self lastSave printString, '"'; cr.

	str crtab; nextPutAll: '"Classes:" ('.
	(encoding at: 3) do: [:sub | str crtab: 2. sub printCompletelyOn: str indent: 3].
	str nextPutAll: ')'; cr.

	str crtab; nextPutAll: '"Class inheritance:" ('.
	(encoding at: 4) do: [:sub | str crtab: 2. sub printCompletelyOn: str indent: 3].
	str nextPutAll: ')'; cr.

	str crtab; nextPutAll: '"Attributes:" ('.
	(encoding at: 5) do: [:sub | str crtab: 2. sub printCompletelyOn: str indent: 3].
	str nextPutAll: ')'; cr.

	str crtab; nextPutAll: '"Class Indices:" ('.
	(encoding at: 6) do: [:sub | str crtab: 2. sub printCompletelyOn: str indent: 3].
	str nextPutAll: ')'; cr.

	str crtab; nextPutAll: '"Root classes:" ('.
	(encoding at: 7) do: [:sub | str crtab: 2. sub printCompletelyOn: str indent: 3].
	str nextPutAll: ')'; cr.

	str crtab; nextPutAll: '"Relationships:" ('.
	(encoding at: 8) do: [:sub | str crtab: 2. sub printCompletelyOn: str indent: 3].
	str nextPutAll: ')'; cr.

	str nextPutAll: ') decodeAsLiteralArray'; cr.

	^str contents!
literalArrayEncoding
	"Convert me to a literal array."

	| theClasses theAttributes theClassIndices theRelationships |
	theClasses := schemaClasses asSortedCollection: [:a :b | a name <= b name].
	theAttributes := OrderedCollection new: 50.
	theClassIndices := OrderedCollection new: 50.
	theRelationships := schemaRelationships asSortedCollection: [:a :b | a name <= b name].
	theClasses do: [:cls |
		cls attributes do: [:attr |
			theAttributes add: (Array with: cls name with: attr literalArrayEncoding)].
		cls indices do: [:ind |
			theClassIndices add: (Array with: cls name with: #objectIndex with: ind literalArrayEncoding)].
	].
	^(Array new: 8)
		at: 1 put: self class fullyQualifiedReference;
		at: 2 put: self lastSave asMilliseconds;
		at: 3 put: (theClasses asArray collect: [:cls |
			cls literalArrayEncoding]);
		at: 4 put: (theClasses asArray collect: [:cls |
			Array with: cls name with: (cls superClass isNil ifTrue: [nil] ifFalse: [cls superClass name])]);
		at: 5 put: theAttributes asArray;
		at: 6 put: theClassIndices asArray;
		at: 7 put: (self rootsCollect: [:r | r type name]) asArray;
		at: 8 put: (theRelationships asArray collect: [:rel |
			rel literalArrayEncoding]);
		yourself!
```

```
!Core.UndefinedObject methodsFor: 'objy helper'!
delete
	"Recursively delete this object from Objectivity.  This nil terminates the recursion (as will identity-like attributes)."

	^self!
!
```

```
Smalltalk defineClass: #Telecom_Account
	superclass: #{Core.Object}
	indexedType: #none
	private: false
	instanceVariableNames: '
		accountNumber <uint64>
		billingAddress <ooShortRef(Telecom_Address)>
		billingMethod <ooShortRef(Telecom_BillingMethod)>
		lines <ooVArray(Telecom_Line)>
		payments <ooVArray(ooShortRef(Telecom_Payment))>'
	classInstanceVariableNames: ''
	imports: ''
	category: 'Generated Code - Applications.GeneratedCodeApp'!


!Telecom_Account class methodsFor: 'Generated - instance creation'!
new
	"*** Generated method ***"

	^super new initializeAccount!
!


!Telecom_Account class methodsFor: 'private: generated'!
ooCodeGenVersion

	^ 1!
ooTypedInstanceVariablesString

	^ '
		accountNumber <uint64>
		billingAddress <ooShortRef(Telecom_Address)>
		billingMethod <ooShortRef(Telecom_BillingMethod)>
		lines <ooVArray(Telecom_Line)>
		payments <ooVArray(ooShortRef(Telecom_Payment))>'!
!


!Telecom_Account methodsFor: 'Generated - comparing'!
= other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	self billingAddress = other billingAddress ifFalse: [^false].
	self accountNumber = other accountNumber ifFalse: [^false].
	self billingMethod = other billingMethod ifFalse: [^false].
	self payments = other payments ifFalse: [^false].
	self lines = other lines ifFalse: [^false].
	^true!
!


!Telecom_Account methodsFor: 'Generated - accessing'!
accountNumber
	"*** Generated method ***"

	accountNumber assertTypeOrNil: Integer.
	^accountNumber!
accountNumber: anInteger
	"*** Generated method ***"

	anInteger assertTypeOrNil: Integer.
	accountNumber := anInteger.
	self changed: #accountNumber.!
addLine: aLine
	"*** Generated method ***"

	"aLine assertType: Telecom_Line."
	lines addElement: aLine.
	self ooUpdate.
	self changed: #lines.!
```

```
addPayment: aPayment
	"*** Generated method ***"

	"aPayment assertType: Telecom_Payment."
	payments addElement: aPayment.
	self ooUpdate.
	self changed: #payments.!
!


!Telecom_Account methodsFor: 'Generated - assignment'!
assignFrom: other
	"*** Generated method ***"

	other == self ifTrue: [^self].
	[other ooClass == self ooClass] assert.
	self billingAddress ooClass == other billingAddress ooClass
		ifTrue: [self billingAddress assignFrom: other billingAddress]
		ifFalse: [
			self billingAddress delete.
			self billingAddress: other billingAddress].
	self accountNumber: other accountNumber.
	self billingMethod ooClass == other billingMethod ooClass
		ifTrue: [self billingMethod assignFrom: other billingMethod]
		ifFalse: [
			self billingMethod delete.
			self billingMethod: other billingMethod].
	self payments = other payments ifFalse: [
		self payments do: [:x | x delete].
		self payments: other payments].
	self lines: other lines.!
!


!Telecom_Account methodsFor: 'Generated - accessing'!
billingAddress
	"*** Generated method ***"

	billingAddress assertTypeOrNil: Telecom_Address.
	^billingAddress!
billingAddress: anAddress
	"*** Generated method ***"

	anAddress assertTypeOrNil: Telecom_Address.
	billingAddress := anAddress.
	self changed: #billingAddress.!
billingMethod
	"*** Generated method ***"

	billingMethod assertTypeOrNil: Telecom_BillingMethod.
	^billingMethod!
billingMethod: aBillingMethod
	"*** Generated method ***"

	aBillingMethod assertTypeOrNil: Telecom_BillingMethod.
	billingMethod := aBillingMethod.
	self changed: #billingMethod.!
clearAccountNumber
	"*** Generated method ***"

	accountNumber := 16rFFFFFFFFFFFFFFFF.
	self changed: #accountNumber.!
clearBillingAddress
	"*** Generated method ***"

	billingAddress delete.
	billingAddress := nil.
	self changed: #billingAddress.!
clearBillingMethod
	"*** Generated method ***"

	billingMethod delete.
	billingMethod := nil.
	self changed: #billingMethod.!
```

```
clearLines
	"*** Generated method ***"

	lines do: [:x | x delete].
	lines replaceWithElements: #().
	self ooUpdate.
	self changed: #lines.!
clearPayments
	"*** Generated method ***"

	payments do: [:x | x delete].
	payments replaceWithElements: #().
	self ooUpdate.
	self changed: #payments.!
!


!Telecom_Account methodsFor: 'Generated - clustering'!
clusterWith: anObjectToClusterWith
	"*** Generated method ***"

	anObjectToClusterWith mioCluster: self.
	billingAddress clusterWith: self.
	billingMethod clusterWith: self.
	payments do: [:x | x clusterWith: self].
	lines do: [:x | x clusterWith: self].!
!


!Telecom_Account methodsFor: 'Generated - deletion'!
delete
	"*** Generated method ***"

	self ooIsPersistent ifFalse: [^self].
	billingAddress delete.
	billingMethod delete.
	payments do: [:x | x delete].
	self ooDelete.!
!


!Telecom_Account methodsFor: 'Generated - comparing'!
hash
	"*** Generated method ***"

	| runningHash |
	runningHash := 0.
	runningHash := (billingAddress goodHash + runningHash) timesRandomMultiplier.
	runningHash := (accountNumber goodHash + runningHash) timesRandomMultiplier.
	runningHash := (billingMethod goodHash + runningHash) timesRandomMultiplier.
	runningHash := ((payments sum: [:x | x goodHash]) + runningHash) timesRandomMultiplier.
	runningHash := ((lines sum: [:x | x goodHash]) + runningHash) timesRandomMultiplier.
	^runningHash bitXor: 16r05E7054E!
hasSameUniquenessKeysAs: other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	^true!
!


!Telecom_Account methodsFor: 'Generated - initialization'!
initializeAccount
	"*** Generated method ***"

	billingAddress := Telecom_Address new.
	accountNumber := 16rFFFFFFFFFFFFFFFF.
	billingMethod := Telecom_BillingMethod new.
	payments := OoVArray new.
	lines := OoVArray new.!
!
```

```
!Telecom_Account methodsFor: 'Generated - accessing'!
lines
	"*** Generated method ***"

	"lines assertAllType: Telecom_Line."
	^lines!
lines: someLines
	"*** Generated method ***"

	"someLines assertAllType: Telecom_Line."
	lines replaceWithElements: someLines.
	self ooUpdate.
	self changed: #lines.!
payments
	"*** Generated method ***"

	"payments assertAllType: Telecom_Payment."
	^payments!
payments: somePayments
	"*** Generated method ***"

	"somePayments assertAllType: Telecom_Payment."
	payments replaceWithElements: somePayments.
	self ooUpdate.
	self changed: #payments.!
!


!Telecom_Account methodsFor: 'Generated - copying'!
postCopy
	"*** Generated method ***"

	super postCopy.
	billingAddress := billingAddress copy.
	billingMethod := billingMethod copy.
	payments := payments collect: [:x | x copy].
	lines := lines copy.!
!


!Telecom_Account methodsFor: 'Generated - accessing'!
removeLine: aLine
	"*** Generated method ***"

	"aLine assertType: Telecom_Line."
	lines removeElement: aLine.
	self ooUpdate.
	self changed: #lines.!
removePayment: aPayment
	"*** Generated method ***"

	"aPayment assertType: Telecom_Payment."
	payments removeElement: aPayment.
	self ooUpdate.
	self changed: #payments.!
!


Smalltalk defineClass: #Telecom_Address
	superclass: #{Core.Object}
	indexedType: #none
	private: false
	instanceVariableNames: '
		city <ooVString>
		country <ooVString>
		postalCode <ooVString>
		state <ooVString>
		street <ooVString>'
	classInstanceVariableNames: ''
	imports: ''
	category: 'Generated Code - Applications.GeneratedCodeApp'!
```

```
!Telecom_Address class methodsFor: 'Generated - instance creation'!
new
	"*** Generated method ***"

	^super new initializeAddress!
!


!Telecom_Address class methodsFor: 'private: generated'!
ooCodeGenVersion

	^ 1!
ooTypedInstanceVariablesString

	^'
		city <ooVString>
		country <ooVString>
		postalCode <ooVString>
		state <ooVString>
		street <ooVString>'!
!


!Telecom_Address methodsFor: 'Generated - comparing'!
= other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	self city = other city ifFalse: [^false].
	self street = other street ifFalse: [^false].
	self country = other country ifFalse: [^false].
	self state = other state ifFalse: [^false].
	self postalCode = other postalCode ifFalse: [^false].
	^true!
!


!Telecom_Address methodsFor: 'Generated - assignment'!
assignFrom: other
	"*** Generated method ***"

	other == self ifTrue: [^self].
	[other ooClass == self ooClass] assert.
	self city: other city.
	self street: other street.
	self country: other country.
	self state: other state.
	self postalCode: other postalCode.!
!


!Telecom_Address methodsFor: 'Generated - accessing'!
city
	"*** Generated method ***"

	city assertTypeOrNil: String.
	^city!
city: aString
	"*** Generated method ***"

	aString assertTypeOrNil: String.
	city := aString.
	self changed: #city.!
clearCity
	"*** Generated method ***"

	city := ''.
	self changed: #city.!
clearCountry
	"*** Generated method ***"

	country := ''.
	self changed: #country.!
```

```
clearPostalCode
	"*** Generated method ***"

	postalCode := ''.
	self changed: #postalCode.!
clearState
	"*** Generated method ***"

	state := ''.
	self changed: #state.!
clearStreet
	"*** Generated method ***"

	street := ''.
	self changed: #street.!
!


!Telecom_Address methodsFor: 'Generated - clustering'!
clusterWith: anObjectToClusterWith
	"*** Generated method ***"

	anObjectToClusterWith mioCluster: self.!
!


!Telecom_Address methodsFor: 'Generated - accessing'!
country
	"*** Generated method ***"

	country assertTypeOrNil: String.
	^country!
country: aString
	"*** Generated method ***"

	aString assertTypeOrNil: String.
	country := aString.
	self changed: #country.!
!


!Telecom_Address methodsFor: 'Generated - deletion'!
delete
	"*** Generated method ***"

	self ooIsPersistent ifFalse: [^self].
	self ooDelete.!
!


!Telecom_Address methodsFor: 'Generated - comparing'!
hash
	"*** Generated method ***"

	| runningHash |
	runningHash := 0.
	runningHash := (city goodHash + runningHash) timesRandomMultiplier.
	runningHash := (street goodHash + runningHash) timesRandomMultiplier.
	runningHash := (country goodHash + runningHash) timesRandomMultiplier.
	runningHash := (state goodHash + runningHash) timesRandomMultiplier.
	runningHash := (postalCode goodHash + runningHash) timesRandomMultiplier.
	^runningHash bitXor: 16r05E7054E!
hasSameUniquenessKeysAs: other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	^true!
!
```

```
!Telecom_Address methodsFor: 'Generated - initialization'!
initializeAddress
	"*** Generated method ***"

	city := ''.
	street := ''.
	country := ''.
	state := ''.
	postalCode := ''.!
!


!Telecom_Address methodsFor: 'Generated - accessing'!
postalCode
	"*** Generated method ***"

	postalCode assertTypeOrNil: String.
	^postalCode!
postalCode: aString
	"*** Generated method ***"

	aString assertTypeOrNil: String.
	postalCode := aString.
	self changed: #postalCode.!
state
	"*** Generated method ***"

	state assertTypeOrNil: String.
	^state!
state: aString
	"*** Generated method ***"

	aString assertTypeOrNil: String.
	state := aString.
	self changed: #state.!
street
	"*** Generated method ***"

	street assertTypeOrNil: String.
	^street!
street: aString
	"*** Generated method ***"

	aString assertTypeOrNil: String.
	street := aString.
	self changed: #street.!
!

"-------------------------------------------------------------------------"
Smalltalk defineClass: #Telecom_BillingMethod
	superclass: #{Core.Object}
	indexedType: #none
	private: false
	instanceVariableNames: ''
	classInstanceVariableNames: ''
	imports: ''
	category: 'Generated Code - Applications.GeneratedCodeApp'!


!Telecom_BillingMethod methodsFor: 'Generated - comparing'!
= other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	^true!
!
```

```
!Telecom_BillingMethod methodsFor: 'Generated - assignment'!
assignFrom: other
	"*** Generated method ***"

	other == self ifTrue: [^self].
	[other ooClass == self ooClass] assert.!
!


!Telecom_BillingMethod methodsFor: 'Generated - clustering'!
clusterWith: anObjectToClusterWith
	"*** Generated method ***"

	anObjectToClusterWith mioCluster: self.!
!


!Telecom_BillingMethod methodsFor: 'Generated - deletion'!
delete
	"*** Generated method ***"

	self ooIsPersistent ifFalse: [^self].
	self ooDelete.!
!


!Telecom_BillingMethod methodsFor: 'Generated - comparing'!
hash
	"*** Generated method ***"

	| runningHash |
	runningHash := 0.
	^runningHash bitXor: 16r05E7054E!
hasSameUniquenessKeysAs: other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	^true!
!

-----------------------
```

```
!Telecom_AutomaticCreditCardBilling methodsFor: 'Generated - deletion'!
delete
	"*** Generated method ***"

	self ooIsPersistent ifFalse: [^self].
	super delete.!
!


!Telecom_AutomaticCreditCardBilling methodsFor: 'Generated - comparing'!
hash
	"*** Generated method ***"

	| runningHash |
	runningHash := super hash.
	^runningHash bitXor: 16r05E7054E!
!

Smalltalk defineClass: #Telecom_MonthlyInvoice
	superclass: #{Telecom_BillingMethod}
	indexedType: #none
	private: false
	instanceVariableNames: ''
	classInstanceVariableNames: ''
	imports: ''
	category: 'Generated Code - Applications.GeneratedCodeApp'!


!Telecom_MonthlyInvoice methodsFor: 'Generated - comparing'!
= other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	super = other ifFalse: [^false].
	^true!
!


!Telecom_MonthlyInvoice methodsFor: 'Generated - assignment'!
assignFrom: other
	"*** Generated method ***"

	other == self ifTrue: [^self].
	[other ooClass == self ooClass] assert.
	super assignFrom: other.!
!


!Telecom_MonthlyInvoice methodsFor: 'Generated - deletion'!
delete
	"*** Generated method ***"

	self ooIsPersistent ifFalse: [^self].
	super delete.!
!


!Telecom_MonthlyInvoice methodsFor: 'Generated - comparing'!
hash
	"*** Generated method ***"

	| runningHash |
	runningHash := super hash.
	^runningHash bitXor: 16r05E7054E!
!
```

```
Smalltalk defineClass: #Telecom_BillingPlan
	superclass: #{Core.Object}
	indexedType: #none
	private: false
	instanceVariableNames: '
		effective <uint64>
		end <uint64>'
	classInstanceVariableNames: ''
	imports: ''
	category: 'Generated Code - Applications.GeneratedCodeApp'!


!Telecom_BillingPlan class methodsFor: 'Generated - instance creation'!
new
	"*** Generated method ***"

	^super new initializeBillingPlan!
!


!Telecom_BillingPlan class methodsFor: 'private: generated'!
ooCodeGenVersion

	^ 1!
ooTypedInstanceVariablesString

	^ '
		effective <uint64>
		end <uint64>'!
!


!Telecom_BillingPlan methodsFor: 'Generated - comparing'!
= other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	self effective = other effective ifFalse: [^false].
	self end = other end ifFalse: [^false].
	^true!
!


!Telecom_BillingPlan methodsFor: 'Generated - assignment'!
assignFrom: other
	"*** Generated method ***"

	other == self ifTrue: [^self].
	[other ooClass == self ooClass] assert.
	self effective: other effective.
	self end: other end.!
!


!Telecom_BillingPlan methodsFor: 'Generated - accessing'!
clearEffective
	"*** Generated method ***"

	effective := 0.
	self changed: #effective.!
clearEnd
	"*** Generated method ***"

	end := 0.
	self changed: #end.!
!


!Telecom_BillingPlan methodsFor: 'Generated - clustering'!
clusterWith: anObjectToClusterWith
	"*** Generated method ***"

	anObjectToClusterWith mioCluster: self.!
!
```

```
!Telecom_BillingPlan methodsFor: 'Generated - deletion'!
delete
	"*** Generated method ***"

	self oolsPersistent ifFalse: [^self].
	self ooDelete.!
!


!Telecom_BillingPlan methodsFor: 'Generated - accessing'!
effective
	"*** Generated method ***"

	(Timestamp fromMilliseconds: effective) assertTypeOrNil: Timestamp.
	^(Timestamp fromMilliseconds: effective)!
effective: aTimestamp
	"*** Generated method ***"

	aTimestamp assertTypeOrNil: Timestamp.
	effective := aTimestamp asMilliseconds.
	self changed: #effective.!
end
	"*** Generated method ***"

	(Timestamp fromMilliseconds: end) assertTypeOrNil: Timestamp.
	^(Timestamp fromMilliseconds: end)!
end: aTimestamp
	"*** Generated method ***"

	aTimestamp assertTypeOrNil: Timestamp.
	end := aTimestamp asMilliseconds.
	self changed: #end.!
!


!Telecom_BillingPlan methodsFor: 'Generated - comparing'!
hash
	"*** Generated method ***"

	| runningHash |
	runningHash := 0.
	runningHash := (effective goodHash + runningHash) timesRandomMultiplier.
	runningHash := (end goodHash + runningHash) timesRandomMultiplier.
	^runningHash bitXor: 16r05E7054E!
hasSameUniquenessKeysAs: other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	^true!
!


!Telecom_BillingPlan methodsFor: 'Generated - initialization'!
initializeBillingPlan
	"*** Generated method ***"

	effective := 0.
	end := 0.!
!


Smalltalk defineClass: #Telecom_FlatRatePlan
	superclass: #{Telecom_BillingPlan}
	indexedType: #none
	private: false
	instanceVariableNames: ''
	classInstanceVariableNames: ''
	imports: ''
	category: 'Generated Code - Applications.GeneratedCodeApp'!
```

```
!Telecom_FlatRatePlan methodsFor: 'Generated - comparing'!
= other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	super = other ifFalse: [^false].
	^true!
!


!Telecom_FlatRatePlan methodsFor: 'Generated - assignment'!
assignFrom: other
	"*** Generated method ***"

	other == self ifTrue: [^self].
	[other ooClass == self ooClass] assert.
	super assignFrom: other.!
!


!Telecom_FlatRatePlan methodsFor: 'Generated - deletion'!
delete
	"*** Generated method ***"

	self ooIsPersistent ifFalse: [^self].
	super delete.!
!


!Telecom_FlatRatePlan methodsFor: 'Generated - comparing'!
hash
	"*** Generated method ***"

	| runningHash |
	runningHash := super hash.
	^runningHash bitXor: 16r05E7054E!
!

Smalltalk defineClass: #Telecom_FriendsAndFamilyPlan
	superclass: #{Telecom_BillingPlan}
	indexedType: #none
	private: false
	instanceVariableNames: ''
	classInstanceVariableNames: ''
	imports: ''
	category: 'Generated Code - Applications.GeneratedCodeApp'!


!Telecom_FriendsAndFamilyPlan methodsFor: 'Generated - comparing'!
= other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	super = other ifFalse: [^false].
	^true!
!


!Telecom_FriendsAndFamilyPlan methodsFor: 'Generated - assignment'!
assignFrom: other
	"*** Generated method ***"

	other == self ifTrue: [^self].
	[other ooClass == self ooClass] assert.
	super assignFrom: other.!
!
```

```
!Telecom_FriendsAndFamilyPlan methodsFor: 'Generated - deletion'!
delete
	"*** Generated method ***"

	self ooIsPersistent ifFalse: [^self].
	super delete.!
!


!Telecom_FriendsAndFamilyPlan methodsFor: 'Generated - comparing'!
hash
	"*** Generated method ***"

	| runningHash |
	runningHash := super hash.
	^runningHash bitXor: 16r05E7054E!
!

Smalltalk defineClass: #Telecom_CallRecord
	superclass: #{Core.Object}
	indexedType: #none
	private: false
	instanceVariableNames: '
		changeNode <ChangeNode>
		indexingState <uint8>
		remoteIndexEntries <ooVArray(IndexEntry)>
		localIndexEntries <ooVArray(ooShortRef(IndexEntry))>
		duration <uint32>
		start <uint64>'
	classInstanceVariableNames: ''
	imports: ''
	category: 'Generated Code - Applications.GeneratedCodeApp'!


!Telecom_CallRecord class methodsFor: 'Generated - instance creation'!
new
	"*** Generated method ***"

	^super new initializeCallRecord!
!


!Telecom_CallRecord class methodsFor: 'private: generated'!
ooCodeGenVersion

	^ 1!
ooTypedInstanceVariablesString

	^'
		changeNode <ChangeNode>
		indexingState <uint8>
		remoteIndexEntries <ooVArray(IndexEntry)>
		localIndexEntries <ooVArray(ooShortRef(IndexEntry))>
		duration <uint32>
		start <uint64>'!
!


!Telecom_CallRecord methodsFor: 'Generated - comparing'!
= other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	self start = other start ifFalse: [^false].
	self duration = other duration ifFalse: [^false].
	^true!
!
```

```
!Telecom_CallRecord methodsFor: 'Generated - assignment'!
assignFrom: other
	"*** Generated method ***"

	other == self ifTrue: [^self].
	[other ooClass == self ooClass] assert.
	self start: other start.
	self duration: other duration.!
!


!Telecom_CallRecord methodsFor: 'Generated - index entries'!
beDeleting
	"*** Generated method ***"

	[self isStable] assert.
	self setIndexingState: (indexingState bitOr: 1).!
beReindexing
	"*** Generated method ***"

	[self isStable] assert.
	self setIndexingState: (indexingState bitOr: 2).!
beStable
	"*** Generated method ***"

	[self isStable not] assert.
	self setIndexingState: (indexingState bitAnd: 3 bitInvert).!
!


!Telecom_CallRecord methodsFor: 'Generated - accessing'!
changeNode
	"*** Generated method ***"

	^changeNode!
changeNode: aChangeNode
	"*** Generated method ***"

	changeNode := aChangeNode.!
clearDuration
	"*** Generated method ***"

	duration := 16rFFFFFFFF.
	self changed: #duration.!
!


!Telecom_CallRecord methodsFor: 'Generated - index entries'!
clearRequests
	"*** Generated method ***"

	self setIndexingState: (indexingState bitAnd: 12 bitInvert).!
!


!Telecom_CallRecord methodsFor: 'Generated - accessing'!
clearStart
	"*** Generated method ***"

	start := 0.
	self changed: #start.!
!


!Telecom_CallRecord methodsFor: 'Generated - clustering'!
clusterWith: anObjectToClusterWith
	"*** Generated method ***"

	anObjectToClusterWith mioCluster: self.!
!
```

```
!Telecom_CallRecord methodsFor: 'Generated - deletion'!
delete
	"*** Generated method ***"

	self ooIsPersistent ifFalse: [^self].
	localIndexEntries do: [:ie | ie delete].
	self ooDelete.!
!


!Telecom_CallRecord methodsFor: 'Generated - accessing'!
duration
	"*** Generated method ***"

	duration assertTypeOrNil: Integer.
	^duration!
duration: anInteger
	"*** Generated method ***"

	anInteger assertTypeOrNil: Integer.
	duration := anInteger.
	self changed: #duration.!
!


!Telecom_CallRecord methodsFor: 'Generated - comparing'!
hash
	"*** Generated method ***"

	| runningHash |
	runningHash := 0.
	runningHash := (start goodHash + runningHash) timesRandomMultiplier.
	runningHash := (duration goodHash + runningHash) timesRandomMultiplier.
	^runningHash bitXor: 16r05E7054E!
!


!Telecom_CallRecord methodsFor: 'Generated - index entries'!
hasRequestedDeletion
	"*** Generated method ***"

	^(indexingState bitAnd: 4) ~= 0!
hasRequestedReindexing
	"*** Generated method ***"

	^(indexingState bitAnd: 8) ~= 0!
!


!Telecom_CallRecord methodsFor: 'Generated - comparing'!
hasSameUniquenessKeysAs: other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	^true!
!


!Telecom_CallRecord methodsFor: 'Generated - index entries'!
indexingState
	"*** Generated method ***"

	^indexingState!
!
```

```
!Telecom_CallRecord methodsFor: 'Generated - initialization'!
initializeCallRecord
	"*** Generated method ***"

	indexingState := 0.
	remoteIndexEntries := OoVArray new.
	localIndexEntries := OoVArray new.
	start := 0.
	duration := 16rFFFFFFFF.!
!


!Telecom_CallRecord methodsFor: 'Generated - index entries'!
isDeleting
	"*** Generated method ***"

	^(indexingState bitAnd: 1) ~= 0!
isReindexing
	"*** Generated method ***"

	^(indexingState bitAnd: 2) ~= 0!
isStable
	"*** Generated method ***"

	^(indexingState bitAnd: 3) = 0!
localIndexEntries
	"*** Generated method ***"

	^localIndexEntries!
localIndexEntries: someIndexEntries
	"*** Generated method ***"

	someIndexEntries assertAllTypeOrNil: IndexEntry.
	localIndexEntries replaceWithElements: someIndexEntries.
	self ooUpdate.!
localOrTransientIndexEntries
	"*** Generated method ***"

	self isStable
		ifTrue: [^self transientIndexEntries]
		ifFalse: [^self localIndexEntries].!
privateSetIndexingState: anInteger
	"*** Generated method ***"

	"Transcript crtab: 5; nextPutAll: 'indexingState: '; print: indexingState; nextPutAll: ' -> '; print: anInteger; nextPutAll: '.'."
	indexingState := anInteger.!
remoteIndexEntries
	"*** Generated method ***"

	^remoteIndexEntries!
remoteIndexEntries: someIndexEntries
	"*** Generated method ***"

	"someIndexEntries assertAllTypeOrNil: IndexEntry." "Too expensive to check"
	remoteIndexEntries replaceWithElements: someIndexEntries.
	self ooUpdate.!
requestDeletion
	"*** Generated method ***"

	[self isStable not] assert.
	self setIndexingState: (indexingState bitOr: 4).!
requestReindexing
	"*** Generated method ***"

	[self isStable not] assert.
	self setIndexingState: (indexingState bitOr: 8).!
setIndexingState: anInteger
	"*** Generated method ***"

	indexingState = anInteger ifFalse: [self privateSetIndexingState: anInteger].!
!
```

```
!Telecom_CallRecord methodsFor: 'Generated - accessing'!
start
	"*** Generated method ***"

	(Timestamp fromMilliseconds: start) assertTypeOrNil: Timestamp.
	^(Timestamp fromMilliseconds: start)!
start: aTimestamp
	"*** Generated method ***"

	aTimestamp assertTypeOrNil: Timestamp.
	start := aTimestamp asMilliseconds.
	self changed: #start.!
!


!Telecom_CallRecord methodsFor: 'Generated - index entries'!
transientIndexEntries
	"*** Generated method ***"

	| transientIndexEntries |
	transientIndexEntries := OrderedCollection new.
	^ transientIndexEntries asArray!
!

Smalltalk defineClass: #Telecom_InboundCallRecord
	superclass: #{Telecom_CallRecord}
	indexedType: #none
	private: false
	instanceVariableNames: '
		callerLine <uint64>'
	classInstanceVariableNames: ''
	imports: ''
	category: 'Generated Code - Applications.GeneratedCodeApp'!


!Telecom_InboundCallRecord class methodsFor: 'Generated - instance creation'!
new
	"*** Generated method ***"

	^super new initializeInboundCallRecord!
!


!Telecom_InboundCallRecord class methodsFor: 'private: generated'!
ooCodeGenVersion

	^ 1!
ooTypedInstanceVariablesString

	^ '
		callerLine <uint64>'!
!


!Telecom_InboundCallRecord methodsFor: 'Generated - comparing'!
= other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	super = other ifFalse: [^false].
	self callerLine = other callerLine ifFalse: [^false].
	^true!
!
```

```
!Telecom_InboundCallRecord methodsFor: 'Generated - assignment'!
assignFrom: other
	"*** Generated method ***"

	other == self ifTrue: [^self].
	[other ooClass == self ooClass] assert.
	super assignFrom: other.
	self callerLine: other callerLine.!
!


!Telecom_InboundCallRecord methodsFor: 'Generated - accessing'!
callerLine
	"*** Generated method ***"

	callerLine assertTypeOrNil: Integer.
	^callerLine!
callerLine: anInteger
	"*** Generated method ***"

	anInteger assertTypeOrNil: Integer.
	callerLine := anInteger.
	self changed: #callerLine.!
clearCallerLine
	"*** Generated method ***"

	callerLine := 16rFFFFFFFFFFFFFFFF.
	self changed: #callerLine.!
!


!Telecom_InboundCallRecord methodsFor: 'Generated - deletion'!
delete
	"*** Generated method ***"

	self ooIsPersistent ifFalse: [^self].
	super delete.!
!


!Telecom_InboundCallRecord methodsFor: 'Generated - comparing'!
hash
	"*** Generated method ***"

	| runningHash |
	runningHash := super hash.
	runningHash := (callerLine goodHash + runningHash) timesRandomMultiplier.
	^runningHash bitXor: 16r05E7054E!
!


!Telecom_InboundCallRecord methodsFor: 'Generated - initialization'!
initializeInboundCallRecord
	"*** Generated method ***"

	callerLine := 16rFFFFFFFFFFFFFFFF.!
!


Smalltalk defineClass: #Telecom_OutboundCallRecord
	superclass: #{Telecom_CallRecord}
	indexedType: #none
	private: false
	instanceVariableNames: '
		calledLine <uint64>'
	classInstanceVariableNames: ''
	imports: ''
	category: 'Generated Code - Applications.GeneratedCodeApp'!
```

```
!Telecom_OutboundCallRecord class methodsFor: 'Generated - instance creation'!
new
	"*** Generated method ***"

	^super new initializeOutboundCallRecord!
!


!Telecom_OutboundCallRecord class methodsFor: 'private: generated'!
ooCodeGenVersion

	^ 1!
ooTypedInstanceVariablesString

	^ '
		calledLine <uint64>'!
!


!Telecom_OutboundCallRecord methodsFor: 'Generated - comparing'!
= other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	super = other ifFalse: [^false].
	self calledLine = other calledLine ifFalse: [^false].
	^true!
!


!Telecom_OutboundCallRecord methodsFor: 'Generated - assignment'!
assignFrom: other
	"*** Generated method ***"

	other == self ifTrue: [^self].
	[other ooClass == self ooClass] assert.
	super assignFrom: other.
	self calledLine: other calledLine.!
!


!Telecom_OutboundCallRecord methodsFor: 'Generated - accessing'!
calledLine
	"*** Generated method ***"

	calledLine assertTypeOrNil: Integer.
	^calledLine!
calledLine: anInteger
	"*** Generated method ***"

	anInteger assertTypeOrNil: Integer.
	calledLine := anInteger.
	self changed: #calledLine.!
clearCalledLine
	"*** Generated method ***"

	calledLine := 16rFFFFFFFFFFFFFFFF.
	self changed: #calledLine.!
!


!Telecom_OutboundCallRecord methodsFor: 'Generated - deletion'!
delete
	"*** Generated method ***"

	self ooIsPersistent ifFalse: [^self].
	super delete.!
!
```

```
!Telecom_OutboundCallRecord methodsFor: 'Generated - comparing'!
hash
	"*** Generated method ***"

	| runningHash |
	runningHash := super hash.
	runningHash := (calledLine goodHash + runningHash) timesRandomMultiplier.
	^runningHash bitXor: 16r05E7054E!
!


!Telecom_OutboundCallRecord methodsFor: 'Generated - initialization'!
initializeOutboundCallRecord
	"*** Generated method ***"

	calledLine := 16rFFFFFFFFFFFFFFFF.!
!


Smalltalk defineClass: #Telecom_ContactChannel
	superclass: #{Core.Object}
	indexedType: #none
	private: false
	instanceVariableNames: ''
	classInstanceVariableNames: ''
	imports: ''
	category: 'Generated Code - Applications.GeneratedCodeApp'!


!Telecom_ContactChannel methodsFor: 'Generated - comparing'!
= other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	^true!
!


!Telecom_ContactChannel methodsFor: 'Generated - assignment'!
assignFrom: other
	"*** Generated method ***"

	other == self ifTrue: [^self].
	[other ooClass == self ooClass] assert.!
!


!Telecom_ContactChannel methodsFor: 'Generated - clustering'!
clusterWith: anObjectToClusterWith
	"*** Generated method ***"

	anObjectToClusterWith mioCluster: self.!
!


!Telecom_ContactChannel methodsFor: 'Generated - deletion'!
delete
	"*** Generated method ***"

	self ooIsPersistent ifFalse: [^self].
	self ooDelete.!
!
```

```
!Telecom_ContactChannel methodsFor: 'Generated - comparing'!
hash
	"*** Generated method ***"

	| runningHash |
	runningHash := 0.
	^runningHash bitXor: 16r05E7054E!
hasSameUniquenessKeysAs: other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	^true!
!

Smalltalk defineClass: #Telecom_CallCenter
	superclass: #{Telecom_ContactChannel}
	indexedType: #none
	private: false
	instanceVariableNames: ''
	classInstanceVariableNames: ''
	imports: ''
	category: 'Generated Code - Applications.GeneratedCodeApp'!

!Telecom_CallCenter methodsFor: 'Generated - comparing'!
= other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	super = other ifFalse: [^false].
	^true!
!

!Telecom_CallCenter methodsFor: 'Generated - assignment'!
assignFrom: other
	"*** Generated method ***"

	other == self ifTrue: [^self].
	[other ooClass == self ooClass] assert.
	super assignFrom: other.!
!

!Telecom_CallCenter methodsFor: 'Generated - deletion'!
delete
	"*** Generated method ***"

	self ooIsPersistent ifFalse: [^self].
	super delete.!
!

!Telecom_CallCenter methodsFor: 'Generated - comparing'!
hash
	"*** Generated method ***"

	| runningHash |
	runningHash := super hash.
	^runningHash bitXor: 16r05E7054E!
!

Smalltalk defineClass: #Telecom_FaceToFace
	superclass: #{Telecom_ContactChannel}
	indexedType: #none
	private: false
	instanceVariableNames: ''
	classInstanceVariableNames: ''
	imports: ''
	category: 'Generated Code - Applications.GeneratedCodeApp'!
```

```
!Telecom_FaceToFace methodsFor: 'Generated - comparing'!
= other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	super = other ifFalse: [^false].
	^true!
!


!Telecom_FaceToFace methodsFor: 'Generated - assignment'!
assignFrom: other
	"*** Generated method ***"

	other == self ifTrue: [^self].
	[other ooClass == self ooClass] assert.
	super assignFrom: other.!
!


!Telecom_FaceToFace methodsFor: 'Generated - deletion'!
delete
	"*** Generated method ***"

	self ooIsPersistent ifFalse: [^self].
	super delete.!
!


!Telecom_FaceToFace methodsFor: 'Generated - comparing'!
hash
	"*** Generated method ***"

	| runningHash |
	runningHash := super hash.
	^runningHash bitXor: 16r05E7054E!
!

Smalltalk defineClass: #Telecom_Internet
	superclass: #{Telecom_ContactChannel}
	indexedType: #none
	private: false
	instanceVariableNames: ''
	classInstanceVariableNames: ''
	imports: ''
	category: 'Generated Code - Applications.GeneratedCodeApp'!


!Telecom_Internet methodsFor: 'Generated - comparing'!
= other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	super = other ifFalse: [^false].
	^true!
!


!Telecom_Internet methodsFor: 'Generated - assignment'!
assignFrom: other
	"*** Generated method ***"

	other == self ifTrue: [^self].
	[other ooClass == self ooClass] assert.
	super assignFrom: other.!
!
```

```
!Telecom_Internet methodsFor: 'Generated - deletion'!
delete
	"*** Generated method ***"

	self ooIsPersistent ifFalse: [^self].
	super delete.!
!


!Telecom_Internet methodsFor: 'Generated - comparing'!
hash
	"*** Generated method ***"

	| runningHash |
	runningHash := super hash.
	^runningHash bitXor: 16r05E7054E!
!

Smalltalk defineClass: #Telecom_Mail
	superclass: #{Telecom_ContactChannel}
	indexedType: #none
	private: false
	instanceVariableNames: ''
	classInstanceVariableNames: ''
	imports: ''
	category: 'Generated Code - Applications.GeneratedCodeApp'!


!Telecom_Mail methodsFor: 'Generated - comparing'!
= other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	super = other ifFalse: [^false].
	^true!
!


!Telecom_Mail methodsFor: 'Generated - assignment'!
assignFrom: other
	"*** Generated method ***"

	other == self ifTrue: [^self].
	[other ooClass == self ooClass] assert.
	super assignFrom: other.!
!


!Telecom_Mail methodsFor: 'Generated - deletion'!
delete
	"*** Generated method ***"

	self ooIsPersistent ifFalse: [^self].
	super delete.!
!


!Telecom_Mail methodsFor: 'Generated - comparing'!
hash
	"*** Generated method ***"

	| runningHash |
	runningHash := super hash.
	^runningHash bitXor: 16r05E7054E!
!
```

```
Smalltalk defineClass: #Telecom_ContactEvent
	superclass: #{Core.Object}
	indexedType: #none
	private: false
	instanceVariableNames: '
		channel <ooShortRef(Telecom_ContactChannel)>'
	classInstanceVariableNames: ''
	imports: ''
	category: 'Generated Code - Applications.GeneratedCodeApp'!


!Telecom_ContactEvent class methodsFor: 'Generated - instance creation'!
new
	"*** Generated method ***"

	^super new initializeContactEvent!
!


!Telecom_ContactEvent class methodsFor: 'private: generated'!
ooCodeGenVersion

	^ 1!
ooTypedInstanceVariablesString

	^'
		channel <ooShortRef(Telecom_ContactChannel)>'!
!


!Telecom_ContactEvent methodsFor: 'Generated - comparing'!
= other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	self channel = other channel ifFalse: [^false].
	^true!
!


!Telecom_ContactEvent methodsFor: 'Generated - assignment'!
assignFrom: other
	"*** Generated method ***"

	other == self ifTrue: [^self].
	[other ooClass == self ooClass] assert.
	self channel ooClass == other channel ooClass
		ifTrue: [self channel assignFrom: other channel]
		ifFalse: [
			self channel delete.
			self channel: other channel].!
!


!Telecom_ContactEvent methodsFor: 'Generated - accessing'!
channel
	"*** Generated method ***"

	channel assertTypeOrNil: Telecom_ContactChannel.
	^channel!
channel: aContactChannel
	"*** Generated method ***"

	aContactChannel assertTypeOrNil: Telecom_ContactChannel.
	channel := aContactChannel.
	self changed: #channel.!
clearChannel
	"*** Generated method ***"

	channel delete.
	channel := nil.
	self changed: #channel.!
!
```

```
!Telecom_ContactEvent methodsFor: 'Generated - clustering'!
clusterWith: anObjectToClusterWith
	"*** Generated method ***"

	anObjectToClusterWith mioCluster: self.
	channel clusterWith: self.!
!


!Telecom_ContactEvent methodsFor: 'Generated - deletion'!
delete
	"*** Generated method ***"

	self ooIsPersistent ifFalse: [^self].
	channel delete.
	self ooDelete.!
!


!Telecom_ContactEvent methodsFor: 'Generated - comparing'!
hash
	"*** Generated method ***"

	| runningHash |
	runningHash := 0.
	runningHash := (channel goodHash + runningHash) timesRandomMultiplier.
	^runningHash bitXor: 16r05E7054E!
hasSameUniquenessKeysAs: other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	^true!
!


!Telecom_ContactEvent methodsFor: 'Generated - initialization'!
initializeContactEvent
	"*** Generated method ***"

	channel := Telecom_ContactChannel new.!
!


!Telecom_ContactEvent methodsFor: 'Generated - copying'!
postCopy
	"*** Generated method ***"

	super postCopy.
	channel := channel copy.!
!


Smalltalk defineClass: #Telecom_ChangeAddress
	superclass: #{Telecom_ContactEvent}
	indexedType: #none
	private: false
	instanceVariableNames: '
		newAddress <ooShortRef(Telecom_Address)>'
	classInstanceVariableNames: ''
	imports: ''
	category: 'Generated Code - Applications.GeneratedCodeApp'!


!Telecom_ChangeAddress class methodsFor: 'Generated - instance creation'!
new
	"*** Generated method ***"

	^super new initializeChangeAddress!
!
```

```
!Telecom_ChangeAddress class methodsFor: 'private: generated'!
ooCodeGenVersion

    ^ 1!
ooTypedInstanceVariablesString

    ^'
        newAddress <ooShortRef(Telecom_Address)>'!
!


!Telecom_ChangeAddress methodsFor: 'Generated - comparing'!
= other
    "*** Generated method ***"

    other == self ifTrue: [^true].
    other ooClass == self ooClass ifFalse: [^false].
    super = other ifFalse: [^false].
    self newAddress = other newAddress ifFalse: [^false].
    ^true!
!


!Telecom_ChangeAddress methodsFor: 'Generated - assignment'!
assignFrom: other
    "*** Generated method ***"

    other == self ifTrue: [^self].
    [other ooClass == self ooClass] assert.
    super assignFrom: other.
    self newAddress ooClass == other newAddress ooClass
        ifTrue: [self newAddress assignFrom: other newAddress]
        ifFalse: [
            self newAddress delete.
            self newAddress: other newAddress].!
!


!Telecom_ChangeAddress methodsFor: 'Generated - accessing'!
clearNewAddress
    "*** Generated method ***"

    newAddress delete.
    newAddress := nil.
    self changed: #newAddress.!
!


!Telecom_ChangeAddress methodsFor: 'Generated - clustering'!
clusterWith: anObjectToClusterWith
    "*** Generated method ***"

    super clusterWith: anObjectToClusterWith.
    newAddress clusterWith: self.!
!


!Telecom_ChangeAddress methodsFor: 'Generated - deletion'!
delete
    "*** Generated method ***"

    self ooIsPersistent ifFalse: [^self].
    newAddress delete.
    super delete.!
!
```

```
!Telecom_ChangeAddress methodsFor: 'Generated - comparing'!
hash
	"*** Generated method ***"

	| runningHash |
	runningHash := super hash.
	runningHash := (newAddress goodHash + runningHash) timesRandomMultiplier.
	^runningHash bitXor: 16r05E7054E!
!


!Telecom_ChangeAddress methodsFor: 'Generated - initialization'!
initializeChangeAddress
	"*** Generated method ***"

	newAddress := Telecom_Address new.!
!


!Telecom_ChangeAddress methodsFor: 'Generated - accessing'!
newAddress
	"*** Generated method ***"

	newAddress assertTypeOrNil: Telecom_Address.
	^newAddress!
newAddress: anAddress
	"*** Generated method ***"

	anAddress assertTypeOrNil: Telecom_Address.
	newAddress := anAddress.
	self changed: #newAddress.!
!


!Telecom_ChangeAddress methodsFor: 'Generated - copying'!
postCopy
	"*** Generated method ***"

	super postCopy.
	newAddress := newAddress copy.!
!
```

```
Smalltalk defineClass: #Telecom_ChangeName
	superclass: #{Telecom_ContactEvent}
	indexedType: #none
	private: false
	instanceVariableNames: '
		newName <ooShortRef(Telecom_Name)>'
	classInstanceVariableNames: ''
	imports: ''
	category: 'Generated Code - Applications.GeneratedCodeApp'!


!Telecom_ChangeName class methodsFor: 'Generated - instance creation'!
new
	"*** Generated method ***"

	^super new initializeChangeName!
!


!Telecom_ChangeName class methodsFor: 'private: generated'!
ooCodeGenVersion

	^1!
ooTypedInstanceVariablesString

	^'
		newName <ooShortRef(Telecom_Name)>'!
!
```

```
!Telecom_ChangeName methodsFor: 'Generated - comparing'!
= other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	super = other ifFalse: [^false].
	self newName = other newName ifFalse: [^false].
	^true!
!


!Telecom_ChangeName methodsFor: 'Generated - assignment'!
assignFrom: other
	"*** Generated method ***"

	other == self ifTrue: [^self].
	[other ooClass == self ooClass] assert.
	super assignFrom: other.
	self newName ooClass == other newName ooClass
		ifTrue: [self newName assignFrom: other newName]
		ifFalse: [
			self newName delete.
			self newName: other newName].!
!


!Telecom_ChangeName methodsFor: 'Generated - accessing'!
clearNewName
	"*** Generated method ***"

	newName delete.
	newName := nil.
	self changed: #newName.!
!


!Telecom_ChangeName methodsFor: 'Generated - clustering'!
clusterWith: anObjectToClusterWith
	"*** Generated method ***"

	super clusterWith: anObjectToClusterWith.
	newName clusterWith: self.!
!


!Telecom_ChangeName methodsFor: 'Generated - deletion'!
delete
	"*** Generated method ***"

	self ooIsPersistent ifFalse: [^self].
	newName delete.
	super delete.!
!


!Telecom_ChangeName methodsFor: 'Generated - comparing'!
hash
	"*** Generated method ***"

	| runningHash |
	runningHash := super hash.
	runningHash := (newName goodHash + runningHash) timesRandomMultiplier.
	^runningHash bitXor: 16r05E7054E!
!


!Telecom_ChangeName methodsFor: 'Generated - initialization'!
initializeChangeName
	"*** Generated method ***"

	newName := Telecom_Name new.!
!
```

```
!Telecom_ChangeName methodsFor: 'Generated - accessing'!
newName
	"*** Generated method ***"

	newName assertTypeOrNil: Telecom_Name.
	^newName!
newName: aName
	"*** Generated method ***"

	aName assertTypeOrNil: Telecom_Name.
	newName := aName.
	self changed: #newName.!
!


!Telecom_ChangeName methodsFor: 'Generated - copying'!
postCopy
	"*** Generated method ***"

	super postCopy.
	newName := newName copy.!
!

---

Smalltalk defineClass: #Telecom_ChangePlan
	superclass: #{Telecom_ContactEvent}
	indexedType: #none
	private: false
	instanceVariableNames: "
	classInstanceVariableNames: "
	imports: "
	category: 'Generated Code - Applications.GeneratedCodeApp'!


!Telecom_ChangePlan methodsFor: 'Generated - comparing'!
= other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	super = other ifFalse: [^false].
	^true!
!


!Telecom_ChangePlan methodsFor: 'Generated - assignment'!
assignFrom: other
	"*** Generated method ***"

	other == self ifTrue: [^self].
	[other ooClass == self ooClass] assert.
	super assignFrom: other.!
!


!Telecom_ChangePlan methodsFor: 'Generated - deletion'!
delete
	"*** Generated method ***"

	self ooIsPersistent ifFalse: [^self].
	super delete.!
!


!Telecom_ChangePlan methodsFor: 'Generated - comparing'!
hash
	"*** Generated method ***"

	| runningHash |
	runningHash := super hash.
	^runningHash bitXor: 16r05E7054E!
!
```

```
Smalltalk defineClass: #Telecom_NewLine
	superclass: #{Telecom_ContactEvent}
	indexedType: #none
	private: false
	instanceVariableNames: ''
	classInstanceVariableNames: ''
	imports: ''
	category: 'Generated Code - Applications.GeneratedCodeApp'!


!Telecom_NewLine methodsFor: 'Generated - comparing'!
= other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	super = other ifFalse: [^false].
	^true!
!


!Telecom_NewLine methodsFor: 'Generated - assignment'!
assignFrom: other
	"*** Generated method ***"

	other == self ifTrue: [^self].
	[other ooClass == self ooClass] assert.
	super assignFrom: other.!
!


!Telecom_NewLine methodsFor: 'Generated - deletion'!
delete
	"*** Generated method ***"

	self ooIsPersistent ifFalse: [^self].
	super delete.!
!


!Telecom_NewLine methodsFor: 'Generated - comparing'!
hash
	"*** Generated method ***"

	| runningHash |
	runningHash := super hash.
	^runningHash bitXor: 16r05E7054E!
!


Smalltalk defineClass: #Telecom_Line_byNumber
	superclass: #{IndexEntry}
	indexedType: #none
	private: false
	instanceVariableNames: '
		number <uint64>'
	classInstanceVariableNames: ''
	imports: ''
	category: 'Generated Code - Applications.GeneratedCodeApp'!


!Telecom_Line_byNumber class methodsFor: 'Generated'!
indexEntriesFor: aLine
	"*** Generated code ***"

	| entries template |
	entries := OrderedCollection new.
	template := self new.
	template indexedObject: aLine.
	template number: aLine number.
	entries add: template copy.
	^entries!
!
```

```
!Telecom_Line_byNumber class methodsFor: 'private: generated'!
ooCodeGenVersion

	^ 1!
ooTypedInstanceVariablesString

	^'
		number <uint64>'!
!


!Telecom_Line_byNumber methodsFor: 'Generated - comparing'!
= other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	self number = other number ifFalse: [^false].
	^true!
!


!Telecom_Line_byNumber methodsFor: 'Generated - assignment'!
assignFrom: other
	"*** Generated code ***"

	number := other number.!
!


!Telecom_Line_byNumber methodsFor: 'Generated - comparing'!
hash
	"*** Generated method ***"

	| runningHash |
	runningHash := 0.
	runningHash := (self number goodHash + runningHash) timesRandomMultiplier.
	^runningHash bitXor: 16r05E7054E!
hashOfKey
	"*** Generated method ***"

	| runningHash |
	runningHash := 0.
	runningHash := (self number goodHash + runningHash) timesRandomMultiplier.
	^runningHash bitXor: 16r05E7054E!
hasSameKeyAs: other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	self number = other number ifFalse: [^false].
	^true!
!


!Telecom_Line_byNumber methodsFor: 'Generated - initialization'!
initialize
	"*** Generated code ***"

	number := 16rFFFFFFFFFFFFFFFF.!
!
```

```
!Telecom_Line_byNumber methodsFor: 'Generated - accessing'!
number
	"*** Generated method ***"

	number assertTypeOrNil: Integer.
	^number!
number: anInteger
	"*** Generated method ***"

	anInteger assertTypeOrNil: Integer.
	number := anInteger.
	self changed: #number.!
!


!Telecom_Line_byNumber methodsFor: 'Generated - searching'!
searchByExample: session
	"*** Generated code ***"

	| clauses predicate |
	clauses := OrderedCollection new.
	number = 16rFFFFFFFFFFFFFFFF ifFalse: [
		clauses add: 'number = ', (SmalltalkInteger convertToPredicateConstant: number)].
	predicate := WriteStream on: (String new: 100).
	clauses
		do: [:clause | predicate nextPutAll: clause]
		separatedBy: [predicate nextPutAll: ' && '].
	^(self targetContainerIn: session)
		scan: self ooClass
		predicate: predicate contents!
!


Smalltalk defineClass: #Telecom_Line
	superclass: #{Core.Object}
	indexedType: #none
	private: false
	instanceVariableNames: '
		calls <ooVArray(Telecom_CallRecord)>
		location <ooShortRef(Telecom_Address)>
		number <uint64>
		plans <ooVArray(ooShortRef(Telecom_BillingPlan))>'
	classInstanceVariableNames: ''
	imports: ''
	category: 'Generated Code - Applications.GeneratedCodeApp'!


!Telecom_Line class methodsFor: 'Generated - instance creation'!
new
	"*** Generated method ***"

	^super new initializeLine!
!


!Telecom_Line class methodsFor: 'private: generated'!
ooCodeGenVersion

	^ 1!
ooTypedInstanceVariablesString

	^'
		calls <ooVArray(Telecom_CallRecord)>
		location <ooShortRef(Telecom_Address)>
		number <uint64>
		plans <ooVArray(ooShortRef(Telecom_BillingPlan))>'!
!
```

```
!Telecom_Line methodsFor: 'Generated - comparing'!
= other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	self number = other number ifFalse: [^false].
	self plans = other plans ifFalse: [^false].
	self location = other location ifFalse: [^false].
	self calls = other calls ifFalse: [^false].
	^true!
!


!Telecom_Line methodsFor: 'Generated - accessing'!
addCall: aCallRecord
	"*** Generated method ***"

	"aCallRecord assertType: Telecom_CallRecord."
	calls addElement: aCallRecord.
	self ooUpdate.
	self changed: #calls.!
addPlan: aBillingPlan
	"*** Generated method ***"

	"aBillingPlan assertType: Telecom_BillingPlan."
	plans addElement: aBillingPlan.
	self ooUpdate.
	self changed: #plans.!
!


!Telecom_Line methodsFor: 'Generated - assignment'!
assignFrom: other
	"*** Generated method ***"

	other == self ifTrue: [^self].
	[other ooClass == self ooClass] assert.
	self number: other number.
	self plans = other plans ifFalse: [
		self plans do: [:x | x delete].
		self plans: other plans].
	self location ooClass == other location ooClass
		ifTrue: [self location assignFrom: other location]
		ifFalse: [
			self location delete.
			self location: other location].
	self calls: other calls.!
!


!Telecom_Line methodsFor: 'Generated - accessing'!
calls
	"*** Generated method ***"

	"calls assertAllType: Telecom_CallRecord."
	^calls!
calls: someCallRecords
	"*** Generated method ***"

	"someCallRecords assertAllType: Telecom_CallRecord."
	calls replaceWithElements: someCallRecords.
	self ooUpdate.
	self changed: #calls.!
clearCalls
	"*** Generated method ***"

	calls do: [:x | x delete].
	calls replaceWithElements: #().
	self ooUpdate.
	self changed: #calls.!
```

```
clearLocation
	"*** Generated method ***"

	location delete.
	location := nil.
	self changed: #location.!
clearNumber
	"*** Generated method ***"

	number := 16rFFFFFFFFFFFFFFFF.
	self changed: #number.!
clearPlans
	"*** Generated method ***"

	plans do: [:x | x delete].
	plans replaceWithElements: #().
	self ooUpdate.
	self changed: #plans.!
!


!Telecom_Line methodsFor: 'Generated - clustering'!
clusterWith: anObjectToClusterWith
	"*** Generated method ***"

	anObjectToClusterWith mioCluster: self.
	plans do: [:x | x clusterWith: self].
	location clusterWith: self.!
!


!Telecom_Line methodsFor: 'Generated - deletion'!
delete
	"*** Generated method ***"

	self ooIsPersistent ifFalse: [^self].
	plans do: [:x | x delete].
	location delete.
	self ooDelete.!
!


!Telecom_Line methodsFor: 'Generated - comparing'!
hash
	"*** Generated method ***"

	| runningHash |
	runningHash := 0.
	runningHash := (number goodHash + runningHash) timesRandomMultiplier.
	runningHash := ((plans sum: [:x | x goodHash]) + runningHash) timesRandomMultiplier.
	runningHash := (location goodHash + runningHash) timesRandomMultiplier.
	runningHash := ((calls sum: [:x | x goodHash]) + runningHash) timesRandomMultiplier.
	^runningHash bitXor: 16r05E7054E!
hasSameUniquenessKeysAs: other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	self number = other number ifFalse: [^false].
	^true!
!


!Telecom_Line methodsFor: 'Generated - initialization'!
initializeLine
	"*** Generated method ***"

	number := 16rFFFFFFFFFFFFFFFF.
	plans := OoVArray new.
	location := Telecom_Address new.
	calls := OoVArray new.!
!
```

```
!Telecom_Line methodsFor: 'Generated - accessing'!
location
	"*** Generated method ***"

	location assertTypeOrNil: Telecom_Address.
	^location!
location: anAddress
	"*** Generated method ***"

	anAddress assertTypeOrNil: Telecom_Address.
	location := anAddress.
	self changed: #location.!
number
	"*** Generated method ***"

	number assertTypeOrNil: Integer.
	^number!
number: anInteger
	"*** Generated method ***"

	anInteger assertTypeOrNil: Integer.
	number := anInteger.
	self changed: #number.!
plans
	"*** Generated method ***"

	"plans assertAllType: Telecom_BillingPlan."
	^plans!
plans: someBillingPlans
	"*** Generated method ***"

	"someBillingPlans assertAllType: Telecom_BillingPlan."
	plans replaceWithElements: someBillingPlans.
	self ooUpdate.
	self changed: #plans.!
!


!Telecom_Line methodsFor: 'Generated - copying'!
postCopy
	"*** Generated method ***"

	super postCopy.
	plans := plans collect: [:x | x copy].
	location := location copy.
	calls := calls copy.!
!


!Telecom_Line methodsFor: 'Generated - accessing'!
removeCall: aCallRecord
	"*** Generated method ***"

	"aCallRecord assertType: Telecom_CallRecord."
	calls removeElement: aCallRecord.
	self ooUpdate.
	self changed: #calls.!
removePlan: aBillingPlan
	"*** Generated method ***"

	"aBillingPlan assertType: Telecom_BillingPlan."
	plans removeElement: aBillingPlan.
	self ooUpdate.
	self changed: #plans.!
!
```

```
Smalltalk defineClass: #Telecom_Name
	superclass: #{Core.Object}
	indexedType: #none
	private: false
	instanceVariableNames: '
		designation <ooVString>
		first <ooVString>
		generation <ooVString>
		last <ooVString>
		middle <ooVString>
		prefix <ooVString>'
	classInstanceVariableNames: ''
	imports: ''
	category: 'Generated Code - Applications.GeneratedCodeApp'!


!Telecom_Name class methodsFor: 'Generated - instance creation'!
new
	"*** Generated method ***"

	^super new initializeName!
!


!Telecom_Name class methodsFor: 'private: generated'!
ooCodeGenVersion

	^ 1!
ooTypedInstanceVariablesString

	^'
		designation <ooVString>
		first <ooVString>
		generation <ooVString>
		last <ooVString>
		middle <ooVString>
		prefix <ooVString>'!
!


!Telecom_Name methodsFor: 'Generated - comparing'!
= other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	self first = other first ifFalse: [^false].
	self last = other last ifFalse: [^false].
	self middle = other middle ifFalse: [^false].
	self prefix = other prefix ifFalse: [^false].
	self generation = other generation ifFalse: [^false].
	self designation = other designation ifFalse: [^false].
	^true!
!


!Telecom_Name methodsFor: 'Generated - assignment'!
assignFrom: other
	"*** Generated method ***"

	other == self ifTrue: [^self].
	[other ooClass == self ooClass] assert.
	self first: other first.
	self last: other last.
	self middle: other middle.
	self prefix: other prefix.
	self generation: other generation.
	self designation: other designation.!
!
```

```
!Telecom_Name methodsFor: 'Generated - accessing'!
clearDesignation
	"*** Generated method ***"

	designation := ''.
	self changed: #designation.!
clearFirst
	"*** Generated method ***"

	first := ''.
	self changed: #first.!
clearGeneration
	"*** Generated method ***"

	generation := ''.
	self changed: #generation.!
clearLast
	"*** Generated method ***"

	last := ''.
	self changed: #last.!
clearMiddle
	"*** Generated method ***"

	middle := ''.
	self changed: #middle.!
clearPrefix
	"*** Generated method ***"

	prefix := ''.
	self changed: #prefix.!
!


!Telecom_Name methodsFor: 'Generated - clustering'!
clusterWith: anObjectToClusterWith
	"*** Generated method ***"

	anObjectToClusterWith mioCluster: self.!
!


!Telecom_Name methodsFor: 'Generated - deletion'!
delete
	"*** Generated method ***"

	self ooIsPersistent ifFalse: [^self].
	self ooDelete.!
!


!Telecom_Name methodsFor: 'Generated - accessing'!
designation
	"*** Generated method ***"

	designation assertTypeOrNil: String.
	^designation!
designation: aString
	"*** Generated method ***"

	aString assertTypeOrNil: String.
	designation := aString.
	self changed: #designation.!
first
	"*** Generated method ***"

	first assertTypeOrNil: String.
	^first!
first: aString
	"*** Generated method ***"

	aString assertTypeOrNil: String.
	first := aString.
	self changed: #first.!
```

```
generation
	"*** Generated method ***"

	generation assertTypeOrNil: String.
	^generation!
generation: aString
	"*** Generated method ***"

	aString assertTypeOrNil: String.
	generation := aString.
	self changed: #generation.!
!


!Telecom_Name methodsFor: 'Generated - comparing'!
hash
	"*** Generated method ***"

	| runningHash |
	runningHash := 0.
	runningHash := (first goodHash + runningHash) timesRandomMultiplier.
	runningHash := (last goodHash + runningHash) timesRandomMultiplier.
	runningHash := (middle goodHash + runningHash) timesRandomMultiplier.
	runningHash := (prefix goodHash + runningHash) timesRandomMultiplier.
	runningHash := (generation goodHash + runningHash) timesRandomMultiplier.
	runningHash := (designation goodHash + runningHash) timesRandomMultiplier.
	^runningHash bitXor: 16r05E7054E!
hasSameUniquenessKeysAs: other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	^true!
!


!Telecom_Name methodsFor: 'Generated - initialization'!
initializeName
	"*** Generated method ***"

	first := ''.
	last := ''.
	middle := ''.
	prefix := ''.
	generation := ''.
	designation := ''.!
!


!Telecom_Name methodsFor: 'Generated - accessing'!
last
	"*** Generated method ***"

	last assertTypeOrNil: String.
	^last!
last: aString
	"*** Generated method ***"

	aString assertTypeOrNil: String.
	last := aString.
	self changed: #last.!
middle
	"*** Generated method ***"

	middle assertTypeOrNil: String.
	^middle!
middle: aString
	"*** Generated method ***"

	aString assertTypeOrNil: String.
	middle := aString.
	self changed: #middle.!
```

```
prefix
	"*** Generated method ***"

	prefix assertTypeOrNil: String.
	^prefix!
prefix: aString
	"*** Generated method ***"

	aString assertTypeOrNil: String.
	prefix := aString.
	self changed: #prefix.!
!

"-------------------------------------------------------------------------"
Smalltalk defineClass: #Telecom_Payment
	superclass: #{Core.Object}
	indexedType: #none
	private: false
	instanceVariableNames: '
		amount <uint32>
		date <uint64>'
	classInstanceVariableNames: ''
	imports: ''
	category: 'Generated Code - Applications.GeneratedCodeApp'!


!Telecom_Payment class methodsFor: 'Generated - instance creation'!
new
	"*** Generated method ***"

	^super new initializePayment!
!


!Telecom_Payment class methodsFor: 'private: generated'!
ooCodeGenVersion

	^ 1!
ooTypedInstanceVariablesString

	^ '
		amount <uint32>
		date <uint64>'!
!


!Telecom_Payment methodsFor: 'Generated - comparing'!
= other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	self date = other date ifFalse: [^false].
	self amount = other amount ifFalse: [^false].
	^true!
!


!Telecom_Payment methodsFor: 'Generated - accessing'!
amount
	"*** Generated method ***"

	amount assertTypeOrNil: Integer.
	^amount!
amount: anInteger
	"*** Generated method ***"

	anInteger assertTypeOrNil: Integer.
	amount := anInteger.
	self changed: #amount.!
!
```

```
!Telecom_Payment methodsFor: 'Generated - assignment'!
assignFrom: other
	"*** Generated method ***"

	other == self ifTrue: [^self].
	[other ooClass == self ooClass] assert.
	self date: other date.
	self amount: other amount.!
!


!Telecom_Payment methodsFor: 'Generated - accessing'!
clearAmount
	"*** Generated method ***"

	amount := 16rFFFFFFFF.
	self changed: #amount.!
clearDate
	"*** Generated method ***"

	date := 0.
	self changed: #date.!
!


!Telecom_Payment methodsFor: 'Generated - clustering'!
clusterWith: anObjectToClusterWith
	"*** Generated method ***"

	anObjectToClusterWith mioCluster: self.!
!


!Telecom_Payment methodsFor: 'Generated - accessing'!
date
	"*** Generated method ***"

	(Timestamp fromMilliseconds: date) assertTypeOrNil: Timestamp.
	^(Timestamp fromMilliseconds: date)!
date: aTimestamp
	"*** Generated method ***"

	aTimestamp assertTypeOrNil: Timestamp.
	date := aTimestamp asMilliseconds.
	self changed: #date.!
!


!Telecom_Payment methodsFor: 'Generated - deletion'!
delete
	"*** Generated method ***"

	self ooIsPersistent ifFalse: [^self].
	self ooDelete.!
!


!Telecom_Payment methodsFor: 'Generated - comparing'!
hash
	"*** Generated method ***"

	| runningHash |
	runningHash := 0.
	runningHash := (date goodHash + runningHash) timesRandomMultiplier.
	runningHash := (amount goodHash + runningHash) timesRandomMultiplier.
	^runningHash bitXor: 16r05E7054E!
hasSameUniquenessKeysAs: other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	^true!
!
```

```
!Telecom_Payment methodsFor: 'Generated - initialization'!
initializePayment
	"*** Generated method ***"

	date := 0.
	amount := 16rFFFFFFFF.!
!


Smalltalk defineClass: #Telecom_Person
	superclass: #{Core.Object}
	indexedType: #none
	private: false
	instanceVariableNames: '
		changeNode <ChangeNode>
		indexingState <uint8>
		remoteIndexEntries <ooVArray(IndexEntry)>
		localIndexEntries <ooVArray(ooShortRef(IndexEntry))>
		accounts <ooVArray(Telecom_Account)>
		addresses <ooVArray(ooShortRef(Telecom_Address))>
		birthDate <uint32>
		contactEvents <ooVArray(Telecom_ContactEvent)>
		email <ooVString>
		name <ooShortRef(Telecom_Name)>
		ssn <uint32>'
	classInstanceVariableNames: ''
	imports: ''
	category: 'Generated Code - Applications.GeneratedCodeApp'!


!Telecom_Person class methodsFor: 'Generated - instance creation'!
new
	"*** Generated method ***"

	^super new initializePerson!
!


!Telecom_Person class methodsFor: 'private: generated'!
ooCodeGenVersion

	^ 1!
ooTypedInstanceVariablesString

	^'
		changeNode <ChangeNode>
		indexingState <uint8>
		remoteIndexEntries <ooVArray(IndexEntry)>
		localIndexEntries <ooVArray(ooShortRef(IndexEntry))>
		accounts <ooVArray(Telecom_Account)>
		addresses <ooVArray(ooShortRef(Telecom_Address))>
		birthDate <uint32>
		contactEvents <ooVArray(Telecom_ContactEvent)>
		email <ooVString>
		name <ooShortRef(Telecom_Name)>
		ssn <uint32>'!
!


!Telecom_Person methodsFor: 'Generated - comparing'!
= other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	self birthDate = other birthDate ifFalse: [^false].
	self email = other email ifFalse: [^false].
	self ssn = other ssn ifFalse: [^false].
	self addresses = other addresses ifFalse: [^false].
	self name = other name ifFalse: [^false].
	self contactEvents = other contactEvents ifFalse: [^false].
	self accounts = other accounts ifFalse: [^false].
	^true!
!
```

```
!Telecom_Person methodsFor: 'Generated - accessing'!
accounts
	"*** Generated method ***"

	"accounts assertAllType: Telecom_Account."
	^accounts!
accounts: someAccounts
	"*** Generated method ***"

	"someAccounts assertAllType: Telecom_Account."
	accounts replaceWithElements: someAccounts.
	self ooUpdate.
	self changed: #accounts.!
addAccount: anAccount
	"*** Generated method ***"

	"anAccount assertType: Telecom_Account."
	accounts addElement: anAccount.
	self ooUpdate.
	self changed: #accounts.!
addAddress: anAddress
	"*** Generated method ***"

	"anAddress assertType: Telecom_Address."
	addresses addElement: anAddress.
	self ooUpdate.
	self changed: #addresses.!
addContactEvent: aContactEvent
	"*** Generated method ***"

	"aContactEvent assertType: Telecom_ContactEvent."
	contactEvents addElement: aContactEvent.
	self ooUpdate.
	self changed: #contactEvents.!
addresses
	"*** Generated method ***"

	"addresses assertAllType: Telecom_Address."
	^addresses!
addresses: someAddresses
	"*** Generated method ***"

	"someAddresses assertAllType: Telecom_Address."
	addresses replaceWithElements: someAddresses.
	self ooUpdate.
	self changed: #addresses.!
!


!Telecom_Person methodsFor: 'Generated - assignment'!
assignFrom: other
	"*** Generated method ***"

	other == self ifTrue: [^self].
	[other ooClass == self ooClass] assert.
	self birthDate: other birthDate.
	self email: other email.
	self ssn: other ssn.
	self addresses = other addresses ifFalse: [
		self addresses do: [:x | x delete].
		self addresses: other addresses].
	self name ooClass == other name ooClass
		ifTrue: [self name assignFrom: other name]
		ifFalse: [
			self name delete.
			self name: other name].
	self contactEvents: other contactEvents.
	self accounts: other accounts.!
!
```

```
!Telecom_Person methodsFor: 'Generated - index entries'!
beDeleting
	"*** Generated method ***"

	[self isStable] assert.
	self setIndexingState: (indexingState bitOr: 1).!
beReindexing
	"*** Generated method ***"

	[self isStable] assert.
	self setIndexingState: (indexingState bitOr: 2).!
beStable
	"*** Generated method ***"

	[self isStable not] assert.
	self setIndexingState: (indexingState bitAnd: 3 bitInvert).!
!


!Telecom_Person methodsFor: 'Generated - accessing'!
birthDate
	"*** Generated method ***"

	(Date fromYYYYMMDDInteger: birthDate) assertTypeOrNil: Date.
	^(Date fromYYYYMMDDInteger: birthDate)!
birthDate: aDate
	"*** Generated method ***"

	aDate assertTypeOrNil: Date.
	birthDate := aDate asYYYYMMDDInteger.
	self changed: #birthDate.!
changeNode
	"*** Generated method ***"

	^changeNode!
changeNode: aChangeNode
	"*** Generated method ***"

	changeNode := aChangeNode.!
clearAccounts
	"*** Generated method ***"

	accounts do: [:x | x delete].
	accounts replaceWithElements: #().
	self ooUpdate.
	self changed: #accounts.!
clearAddresses
	"*** Generated method ***"

	addresses do: [:x | x delete].
	addresses replaceWithElements: #().
	self ooUpdate.
	self changed: #addresses.!
clearBirthDate
	"*** Generated method ***"

	birthDate := 0.
	self changed: #birthDate.!
clearContactEvents
	"*** Generated method ***"

	contactEvents do: [:x | x delete].
	contactEvents replaceWithElements: #().
	self ooUpdate.
	self changed: #contactEvents.!
clearEmail
	"*** Generated method ***"

	email := ''.
	self changed: #email.!
```

```
clearName
	"*** Generated method ***"

	name delete.
	name := nil.
	self changed: #name.!
!


!Telecom_Person methodsFor: 'Generated - index entries'!
clearRequests
	"*** Generated method ***"

	self setIndexingState: (indexingState bitAnd: 12 bitInvert).!
!


!Telecom_Person methodsFor: 'Generated - accessing'!
clearSsn
	"*** Generated method ***"

	ssn := 16rFFFFFFFF.
	self changed: #ssn.!
!


!Telecom_Person methodsFor: 'Generated - clustering'!
clusterWith: anObjectToClusterWith
	"*** Generated method ***"

	anObjectToClusterWith mioCluster: self.
	addresses do: [:x | x clusterWith: self].
	name clusterWith: self.
	contactEvents do: [:x | x clusterWith: self].
	accounts do: [:x | x clusterWith: self].!
!


!Telecom_Person methodsFor: 'Generated - accessing'!
contactEvents
	"*** Generated method ***"

	"contactEvents assertAllType: Telecom_ContactEvent."
	^contactEvents!
contactEvents: someContactEvents
	"*** Generated method ***"

	"someContactEvents assertAllType: Telecom_ContactEvent."
	contactEvents replaceWithElements: someContactEvents.
	self ooUpdate.
	self changed: #contactEvents.!
!


!Telecom_Person methodsFor: 'Generated - deletion'!
delete
	"*** Generated method ***"

	self ooIsPersistent ifFalse: [^self].
	addresses do: [:x | x delete].
	name delete.
	localIndexEntries do: [:ie | ie delete].
	self ooDelete.!
!
```

```
!Telecom_Person methodsFor: 'Generated - accessing'!
email
	"*** Generated method ***"

	email assertTypeOrNil: String.
	^email!
email: aString
	"*** Generated method ***"

	aString assertTypeOrNil: String.
	email := aString.
	self changed: #email.!
!


!Telecom_Person methodsFor: 'Generated - comparing'!
hash
	"*** Generated method ***"

	| runningHash |
	runningHash := 0.
	runningHash := (birthDate goodHash + runningHash) timesRandomMultiplier.
	runningHash := (email goodHash + runningHash) timesRandomMultiplier.
	runningHash := (ssn goodHash + runningHash) timesRandomMultiplier.
	runningHash := ((addresses sum: [:x | x goodHash]) + runningHash) timesRandomMultiplier.
	runningHash := (name goodHash + runningHash) timesRandomMultiplier.
	runningHash := ((contactEvents sum: [:x | x goodHash]) + runningHash) timesRandomMultiplier.
	runningHash := ((accounts sum: [:x | x goodHash]) + runningHash) timesRandomMultiplier.
	^runningHash bitXor: 16r05E7054E!
!


!Telecom_Person methodsFor: 'Generated - index entries'!
hasRequestedDeletion
	"*** Generated method ***"

	^(indexingState bitAnd: 4) ~= 0!
hasRequestedReindexing
	"*** Generated method ***"

	^(indexingState bitAnd: 8) ~= 0!
!


!Telecom_Person methodsFor: 'Generated - comparing'!
hasSameUniquenessKeysAs: other
	"*** Generated method ***"

	other == self ifTrue: [^true].
	other ooClass == self ooClass ifFalse: [^false].
	^true!
!


!Telecom_Person methodsFor: 'Generated - index entries'!
indexingState
	"*** Generated method ***"

	^indexingState!
!
```

```
!Telecom_Person methodsFor: 'Generated - initialization'!
initializePerson
	"*** Generated method ***"

	indexingState := 0.
	remoteIndexEntries := OoVArray new.
	localIndexEntries := OoVArray new.
	birthDate := 0.
	email := ''.
	ssn := 16rFFFFFFFF.
	addresses := OoVArray new.
	name := Telecom_Name new.
	contactEvents := OoVArray new.
	accounts := OoVArray new.!
!


!Telecom_Person methodsFor: 'Generated - index entries'!
isDeleting
	"*** Generated method ***"

	^(indexingState bitAnd: 1) ~= 0!
isReindexing
	"*** Generated method ***"

	^(indexingState bitAnd: 2) ~= 0!
isStable
	"*** Generated method ***"

	^(indexingState bitAnd: 3) = 0!
localIndexEntries
	"*** Generated method ***"

	^localIndexEntries!
localIndexEntries: someIndexEntries
	"*** Generated method ***"

	someIndexEntries assertAllTypeOrNil: IndexEntry.
	localIndexEntries replaceWithElements: someIndexEntries.
	self ooUpdate.!
localOrTransientIndexEntries
	"*** Generated method ***"

	self isStable
		ifTrue: [^self transientIndexEntries]
		ifFalse: [^self localIndexEntries].!
!


!Telecom_Person methodsFor: 'Generated - accessing'!
name
	"*** Generated method ***"

	name assertTypeOrNil: Telecom_Name.
	^name!
name: aName
	"*** Generated method ***"

	aName assertTypeOrNil: Telecom_Name.
	name := aName.
	self changed: #name.!
!


!Telecom_Person methodsFor: 'Generated - copying'!
postCopy
	"*** Generated method ***"

	super postCopy.
	addresses := addresses collect: [:x | x copy].
	name := name copy.
	contactEvents := contactEvents copy.
	accounts := accounts copy.!
!
```

```
!Telecom_Person methodsFor: 'Generated - index entries'!
privateSetIndexingState: anInteger
	"*** Generated method ***"

	"Transcript crtab: 5; nextPutAll: 'indexingState: '; print: indexingState; nextPutAll: ' -> '; print: anInteger; nextPutAll: '.'"
	indexingState := anInteger.!
remoteIndexEntries
	"*** Generated method ***"

	^remoteIndexEntries!
remoteIndexEntries: someIndexEntries
	"*** Generated method ***"

	"someIndexEntries assertAllTypeOrNil: IndexEntry." "Too expensive to check"
	remoteIndexEntries replaceWithElements: someIndexEntries.
	self ooUpdate.!
!


!Telecom_Person methodsFor: 'Generated - accessing'!
removeAccount: anAccount
	"*** Generated method ***"

	"anAccount assertType: Telecom_Account."
	accounts removeElement: anAccount.
	self ooUpdate.
	self changed: #accounts.!
removeAddress: anAddress
	"*** Generated method ***"

	"anAddress assertType: Telecom_Address."
	addresses removeElement: anAddress.
	self ooUpdate.
	self changed: #addresses.!
removeContactEvent: aContactEvent
	"*** Generated method ***"

	"aContactEvent assertType: Telecom_ContactEvent."
	contactEvents removeElement: aContactEvent.
	self ooUpdate.
	self changed: #contactEvents.!
!


!Telecom_Person methodsFor: 'Generated - index entries'!
requestDeletion
	"*** Generated method ***"

	[self isStable not] assert.
	self setIndexingState: (indexingState bitOr: 4).!
requestReindexing
	"*** Generated method ***"

	[self isStable not] assert.
	self setIndexingState: (indexingState bitOr: 8).!
setIndexingState: anInteger
	"*** Generated method ***"

	indexingState = anInteger ifFalse: [self privateSetIndexingState: anInteger].!
!


!Telecom_Person methodsFor: 'Generated - accessing'!
ssn
	"*** Generated method ***"

	ssn assertTypeOrNil: Integer.
	^ssn!
ssn: anInteger
	"*** Generated method ***"

	anInteger assertTypeOrNil: Integer.
	ssn := anInteger.
	self changed: #ssn.!
```

```
!

!Telecom_Person methodsFor: 'Generated - index entries'!
transientIndexEntries
	"*** Generated method ***"

	| transientIndexEntries |
	transientIndexEntries := OrderedCollection new.
	^ transientIndexEntries asArray!
!

Smalltalk.Applications defineClass: #GeneratedCodeApp
	superclass: #{ENVY.Application}
	indexedType: #none
	private: false
	instanceVariableNames: ''
	classInstanceVariableNames: ''
	imports: ''
	category: 'ENVY/Manager'!

!Applications.GeneratedCodeApp class methodsFor: 'EM-Internal'!
_PRAGMA_

	"Smalltalk(
		defineNameSpace: #Test
		private: false
		imports: 'private Smalltalk.*'
		category: 'Generated Code - Applications.GeneratedCodeApp')"
	"Smalltalk(
		defineNameSpace: #Telecom
		private: false
		imports: 'private Smalltalk.*'
		category: 'Generated Code - Applications.GeneratedCodeApp')"!
!
```

```
Smalltalk defineClass: #Activator
	superclass: #{Core.Object}
	indexedType: #none
	private: false
	instanceVariableNames: 'bytes bytesIndex classes objects translation classFixups '
	classInstanceVariableNames: ''
	imports: ''
	category: ''!


!Activator class methodsFor: 'private-recognition'!
classRecognizes: aStream
	"Test if the stream's format is recognizeable by this class."

	| savedPosition header |
	savedPosition := aStream position.
	header := self headerString asByteArray.
	1 to: header size do: [:index |
		(aStream atEnd or: [aStream next ~= (header at: index)]) ifTrue: [
			aStream position: savedPosition.
			^false]].
	aStream position: savedPosition.
	^true!
!


!Activator class methodsFor: 'conversion'!
convert: byteArray
	"Answer an object representing the activation of the given byteArray.  See
	 Passivator class>>whatIsTheFileFormat for more details."

	| whichClass byteStream |
	byteArray assertType: ByteArray.
	byteStream := byteArray readStream.
	whichClass := self withAllSubclasses
		detect: [:class | class classRecognizes: byteStream]
		ifNone: [self error: 'Unsupported Activator/Passivator format'].
	^whichClass new convert: byteArray!
!


!Activator class methodsFor: 'private-recognition'!
headerString
	"Answer the prefix used to identify this format."

	^'Passivator_VW1.2'!
!


!Activator class methodsFor: 'conversion'!
recognizes: aStream
	"Test if the stream's format is recognizeable by myself.
	 Always restores the stream's position to what it was when
	 the method was called."

	(self classRecognizes: aStream) ifTrue: [^true].
	^self subclasses any: [:subclass | subclass recognizes: aStream]!
!


!Activator methodsFor: 'private-activation'!
activateObjects
	"Ask each object to do its activation translation.  Activate the objects in
	 bottom-up order (in the absence of relevant cycles) to simplify user fixup
	 code.  The Passivator had the responsibility of writing the objects in
	 reverse-bottom-up order to satisfy this constraint."

	translation := IdentityDictionary new: objects size.
	objects size to: 1 by: -1 do: [:index |
		| before after |
		before := objects at: index.
		self translateFieldsOf: before.
		"Ok, the fields have been translated.  Now activate the current one..."
		after := before postLoadActivation: self.
		before == after ifFalse: [translation at: before put: after]].!
!
```

```
!Activator methodsFor: 'private-converting'!
convert: byteArray
	"Answer anObject created from the description in byteArray."

	| header root |
	byteArray assertType: ByteArray.
	bytes := byteArray.
	bytesIndex := 0.
	header := self class headerString.
	(bytes copyFrom: 1 to: header size) asString = header ifFalse: [self error: 'Invalid header encountered'].
	bytesIndex := bytesIndex + header size.
	self readClasses.
	self readObjects.
	root := self objectFromIndex: self readInt.
	bytesIndex = bytes size ifFalse: [self error: 'Expected end of data'].
	self activateObjects.
	^translation at: root ifAbsent: [root].!
!


!Activator methodsFor: 'private-encoding'!
objectFromIndex: index
	"Answer a non-negative integer representing anObject in the current scheme.
	 See Passivator class>>whatIsTheFileFormat and Passivator>>indexOfObject:
	 for more details."

	| mod4 |
	mod4 := index \\ 4.
	mod4 == 0 ifTrue: [
		index == 0  ifTrue: [^nil].
		index == 4 ifTrue: [^true].
		index == 8 ifTrue: [^false].
		^Character value: (index - 12 bitShift: -2)].
	mod4 == 1 ifTrue: [
		index \\ 8 = 1
			ifTrue: [^1 - index bitShift: -3]
			ifFalse: [^index - 5 bitShift: -3]].
	mod4 == 2 ifTrue: [^classes at: (index - 2 bitShift: -2) + 1].
	"[mod4 == 3] assert."
	^objects at: (index "- 3" bitShift: -2) + 1!
!
```

```
!Activator methodsFor: 'private-reading'!
readClasses

    ^self readClassesStartingAt: 1!
readClassesStartingAt: firstClassIndex

    | howMany oldClasses classIndex |
    howMany := self readInt.
    oldClasses := classes isNil ifTrue: [#()] ifFalse: [classes].
    classes := Array new: howMany.
    classes replaceFrom: 1 to: oldClasses size with: oldClasses startingAt: 1.
    classFixups isNil ifTrue: [
        classFixups := IdentityDictionary new: howMany * 2].
    classIndex := firstClassIndex.
    [classIndex <= howMany] whileTrue: [
        | name class format fileInstVars imageInstVars stream |
        name := self readSizedString.
        (name includes: Character space)
            ifTrue: [
                "It's a metaclass"
                (name readStream upTo: Character space; upToEnd) = 'class' ifFalse: [
                    self error: 'expected metaclass name'].
                class := (name readStream upTo: Character space) asQualifiedReference value class]
            ifFalse: [
                "It's a class"
                class := name asQualifiedReference value].
        classes at: classIndex put: class.
        format := self readInt.
        stream := ReadStream on: self readSizedString.
        fileInstVars := OrderedCollection new: 20.
        [stream atEnd] whileFalse: [fileInstVars addLast: (stream upTo: $ )].
        imageInstVars := class allInstVarNames.
        (format ~= class format or: [fileInstVars ~= imageInstVars]) ifTrue: [
            classFixups at: class put: (class
                activatorFixupForFormat: format
                instVars: fileInstVars
                isVariable: (format bitAnd: Object indexableMask) ~= 0
                isBits: (format bitAnd: Object pointersMask) ~= Object pointersMask)].
        classIndex := classIndex + 1.
    ].!
readInt
    "Read a non-negative integer from the stream.  See Passivator class>>whatIsFileFormat."

"   | byte |
    byte := bytes at: (bytesIndex := bytesIndex + 1).
    byte <= 127 ifTrue: [^byte].
    ^(self readInt bitShift: 7) + (byte bitAnd: 127)  "

    | low high |
    low := bytes at: (bytesIndex := bytesIndex + 1).
    low <= 127 ifTrue: [^low].
    high := bytes at: (bytesIndex := bytesIndex + 1).
    high <= 127
        ifTrue: [^(high bitShift: 7) + low - 128]
        ifFalse: [^(self readInt bitShift: 14) + (high bitShift: 7) + low - 16512  "128*128 + 128"]!
```

```
readObjects

    | howMany class object |
    howMany := self readInt.

    "First pass: Just create the empty (appropriately-sized) objects..."
    objects := Array new: howMany.
    1 to: howMany do: [:i |
        class := classes at: 1 + self readInt.
        objects at: i put: (class isVariable
            ifTrue: [class basicNew: self readInt]
            ifFalse: [class basicNew])].

    "Second pass: Read object contents, setting object references and byte data..."
    1 to: howMany do: [:objectIndex |
        | fixup |
        object := objects at: objectIndex.
        fixup := classFixups at: object class ifAbsent: [nil].
        fixup isNil
            ifTrue: [
                1 to: object class instSize do: [:i |
                    object instVarAt: i put: (self objectFromIndex: self readInt)].
                object class isVariable ifTrue: [
                    object class isBits
                        ifTrue: [self readRawBytesInto: object]
                        ifFalse: [
                            1 to: object basicSize do: [:i |
                                object basicAt: i put: (self objectFromIndex: self readInt)]]]]
            ifFalse: [
                fixup value: object value: self]].!
readRawBytesInto: anObject
    "Fill an object's byte-indexable fields from the stream."

    anObject class isBits ifFalse: [self error: 'Class''s kind is wrong'].
    1 to: anObject basicSize do: [:i |
        anObject basicAt: i put: (bytes at: (bytesIndex := bytesIndex + 1))].!
readSizedString
    "Read a string preceded by its size."

    | size str |
    size := self readInt.
    str := bytes copyFrom: bytesIndex + 1 to: (bytesIndex := bytesIndex + size).
    ^str asString!
!


!Activator methodsFor: 'private-activation'!
translateFieldsOf: anObject
    "Translate the fields of the object, based on current available translations."

    | class |
    translation size = 0 ifTrue: [^self].  "Short circuit if no mappings have occurred yet"
    class := anObject class.
    1 to: class instSize do: [:i |
        | field |
        field := anObject instVarAt: i.
        (translation includesKey: field) ifTrue: [
            anObject instVarAt: i put: (translation at: field ifAbsent: [nil error: 'Invalid translation table'])]].
    (class isVariable and: [class isPointers]) ifTrue: [
        1 to: anObject basicSize do: [:i |
            | field |
            field := anObject basicAt: i.
            (translation includesKey: field) ifTrue: [
                anObject basicAt: i put: (translation at: field ifAbsent: [nil error: 'Invalid translation table'])]]].!
!


Smalltalk defineClass: #ContinuousActivator
    superclass: #{Activator}
    indexedType: #none
    private: false
    instanceVariableNames: ''
    classInstanceVariableNames: ''
    imports: ''
    category: ''!
```

```
!ContinuousActivator class methodsFor: 'accessing'!
headerString
	"Answer the prefix used to identify this format."

	^'CP5i'!
!


!ContinuousActivator methodsFor: 'private-converting'!
convert: byteArray
	"Answer anObject created from the description in byteArray."

	| result |
	result := super convert: byteArray.
	objects := bytes := translation := nil.  "Save some memory"
	^result!
!


!ContinuousActivator methodsFor: 'private-reading'!
readClasses

	^self readClassesStartingAt: (classes isNil ifTrue: [1] ifFalse: [classes size + 1])!
!

Smalltalk defineClass: #MiosoftActivator
	superclass: #{ContinuousActivator}
	indexedType: #none
	private: false
	instanceVariableNames: 'session '
	classInstanceVariableNames: ''
	imports: ''
	category: '!


!MiosoftActivator class methodsFor: 'accessing'!
headerString
	"Answer the prefix used to identify this format."

	^'M'	"M is for Miosoft."!
!


!MiosoftActivator methodsFor: 'accessing'!
session

	^session!
session: anOoSession

	session := anOoSession.!
!

Smalltalk defineClass: #PassivationLargeIdentityDictionary
	superclass: #{Core.Collection}
	indexedType: #none
	private: false
	instanceVariableNames: 'byCategory '
	classInstanceVariableNames: ''
	imports: ''
	category: '!


!PassivationLargeIdentityDictionary class methodsFor: 'instance creation'!
new

	^super new initialize!
!
```

```
!PassivationLargeIdentityDictionary methodsFor: 'As yet unclassified'!
at: key
	"Look up the object at key."

	| map |
	map := byCategory at: (self primaryCategorizationOf: key).
	^map at: key!
at: key ifAbsent: aBlock
	"Look up the object at key.  If unknown, answer aBlock's value."

	| map |
	map := byCategory at: (self primaryCategorizationOf: key) ifAbsent: [nil].
	map isNil ifTrue: [^aBlock value].
	^map at: key ifAbsent: aBlock!
at: key ifAbsentPut: aBlock
	"Look up the object at key.  If unknown, store aBlock's value under that key and answer that value."

	| map |
	map := byCategory at: (self primaryCategorizationOf: key) ifAbsentPut: [IdentityDictionary new].
	^map at: key ifAbsentPut: aBlock!
at: key put: value
	"Store value at key."

	| category map |
	category := self primaryCategorizationOf: key.
	map := byCategory at: category ifAbsent: [byCategory at: category put: IdentityDictionary new].
	^map at: key put: value!
do: aBlock
	"Execute aBlock with each value."

	byCategory do: [:map |
		map do: aBlock].!
includesKey: key
	"Look up the key.  Answer whether a value was found."

	| map |
	map := byCategory at: (self primaryCategorizationOf: key) ifAbsent: [^false].
	^map includesKey: key!
initialize

	byCategory := IdentityDictionary new: 20.!
keysAndValuesDo: twoArgBlock
	"Execute twoArgBlock with each key and value."

	byCategory do: [:map |
		map keysAndValuesDo: twoArgBlock].!
keysDo: aBlock
	"Execute aBlock with each key."

	byCategory do: [:map |
		map keysDo: aBlock].!
primaryCategorizationOf: anObject
	"Answer something permanent about the object, besides its identity hash, on which to initially segregate
	 objects (to supplement the pathologically bad 14-bit identity hashing VisualWorks forces."

	^anObject oolsPersistent
		ifTrue: [anObject oid containerNumber]
		ifFalse: [anObject class]!
size

	| s |
	s := 0.
	byCategory do: [:map | s := s + map size].
	^s!
!
```

```
Smalltalk defineClass: #PrivatePassivationByteArray
	superclass: #{Core.ByteArray}
	indexedType: #bytes
	private: false
	instanceVariableNames: ''
	classInstanceVariableNames: ''
	imports: ''
	category: ''!


!PrivatePassivationByteArray methodsFor: 'accessing'!
at: index put: byteValue
	"Store the argument value in the indexable field of the receiver
	indicated by index. Fail if the index is not a SmallInteger or
	is out of bounds, or if the byteValue is not a SmallInteger
	between 0 and 255. Answer the value that was stored. If the
	store is beyond the end of the receiver, grow it to make room,
	and a linear amount of excess, to offset the cost of copying
	the data. This keeps amortized growth cost linear."

	<primitive: 421>
	index > self size ifTrue: [self growTo: index * 4 + 1000].
	^super at: index put: byteValue!
growTo: newSize
	"Grow the receiver to the given new size. Do a become: to enlarge it in place.
	 Answer nil."

	| newMe |
	newMe := self class new: newSize.
	newMe replaceBytesFrom: 1 to: self size with: self startingAt: 1.
	newMe become: self.
	"Answer nil to avoid receiver interpretation problems."
	^nil!
replaceBytesFrom: start to: stop with: replacement startingAt: repStart
	"This destructively replaces elements from start to stop in the receiver
	starting at index, repStart, in the collection, replacement. Answer the
	receiver. If the transfer is beyond the end of the receiver, grow it to
	make room, and a linear amount of excess, to offset the cost of copying
	the data. This keeps amortized growth cost linear."

	<primitive: 559>
	stop > self size ifTrue: [self growTo: stop * 4 + 1000].
	^super replaceBytesFrom: start to: stop with: replacement startingAt: repStart!
!


Smalltalk defineClass: #PassivatedPersistentObjectReference
	superclass: #{Core.Object}
	indexedType: #bytes
	private: false
	instanceVariableNames: ''
	classInstanceVariableNames: ''
	imports: ''
	category: ''!


!PassivatedPersistentObjectReference class methodsFor: 'instance creation'!
new

	^super new: 8!
!
```

```
!PassivatedPersistentObjectReference methodsFor: 'accessing'!
oid: anOid

    | con obj |
    con := anOid containerNumber.
    self basicAt: 1 put: (con bitShift: -24).
    self basicAt: 2 put: ((con bitShift: -16) bitAnd: 255).
    self basicAt: 3 put: ((con bitShift: -8) bitAnd: 255).
    self basicAt: 4 put: (con bitAnd: 255).

    obj := anOid objectNumber.
    self basicAt: 5 put: (obj bitShift: -24).
    self basicAt: 6 put: ((obj bitShift: -16) bitAnd: 255).
    self basicAt: 7 put: ((obj bitShift: -8) bitAnd: 255).
    self basicAt: 8 put: (obj bitAnd: 255).!
!


!PassivatedPersistentObjectReference methodsFor: 'passivation/activation'!
postLoadActivation: anActivator
    "Activate myself now that I have been loaded by anActivator.  Answer the object
     to substitute for myself in parent objects (when not involved in cycles)."

    | con obj container object |
    anActivator assertType: MiosoftActivator.
    anActivator session assertType: OoSession.

    con := (self basicAt: 4)
        + ((self basicAt: 3) bitShift: 8)
        + ((self basicAt: 2) bitShift: 16)
        + ((self basicAt: 1) bitShift: 24).
    obj := (self basicAt: 8)
        + ((self basicAt: 7) bitShift: 8)
        + ((self basicAt: 6) bitShift: 16)
        + ((self basicAt: 5) bitShift: 24).

    container := anActivator session ooProvideContainerFor: con with: obj.
    object := container ooProvideObject: obj.
    ^object!
!

-----------------------
```

```
!PassivationRepresentative methodsFor: 'private-accessing'!
decrementParentCount
	"Answers the resulting count."

	^ parentCount := parentCount - 1!
incrementParentCount

	parentCount := parentCount + 1.!
index

	^index!
index: anInteger

	index := anInteger.!
object

	^object!
object: anObject

"	[anObject notNil] assert."
	object := anObject.!
parentCount

	^parentCount!
parentCount: anInteger

	parentCount := anInteger!
!


!PassivationRepresentative methodsFor: 'private-writing'!
passivationIntegerEncodingIn: aPassivator

	^index * 4 + 3!
!


!PassivationRepresentative methodsFor: 'private-accessing'!
subrepresentatives

	^subrepresentatives!
!


!PassivationRepresentative methodsFor: 'private-writing'!
writePart1For: passivator
	"Write the first half of my local data.  I always write my class's id and my
	basicSize (if variable) in the first pass so the object shells can be constructed
	at load time."

	passivator writeInt: (passivator indexOfClass: object class).
	object class isVariable ifTrue: [
		passivator writeInt: object basicSize].!
writePart2For: passivator
	"Write the final half of my local data.  See writePart1For:.  I always write my named
	instance variables, then each of my indexed (pointer) instance variables (if any)."

	1 to: subrepresentatives size do: [:i |
		passivator writeReferenceToRepresentative: (subrepresentatives at: i)].
	object class isBits ifTrue: [
		"Write out the object's bytes."
		passivator writeRawBytes: object].!
!


Smalltalk defineClass: #Passivator
	superclass: #{Core.Object}
	indexedType: #none
	private: false
	instanceVariableNames: 'bytes bytesIndex objectToRepresentative allRepresentatives stream root classesToIndex stack '
	classInstanceVariableNames: ''
	imports: ''
	category: ''!
```

```
!Passivator class methodsFor: 'instance creation'!
convert: anObject
	"Answer a ByteArray representing the passivation of anObject.  For example,
	'Passivator convert: (1/2@3 corner: 4@5)'.  To turn this ByteArray into a copy
	of anObject, run 'Activator convert: aByteArray'."

	^self new convert: anObject!
!


!Passivator class methodsFor: 'accessing'!
headerString
	"Answer the prefix used to identify this format."

	^'Passivator_VW1.2'!
!


!Passivator class methodsFor: 'documentation'!
whatIsTheFileFormat
	"
Ints (positive integers often representing something else) are written low-to-high, 7 bits at a time.  A byte in the range 128..255 indicates there is more data following, and that the low seven bits contribute to the int being formed.  A byte in the range 0..127 is the int itself.  For example, the stream 130,99,... is parsed as:  Read byte (130).  It's > 127 so read an int from what's left, multiply by 128, and add 130-128.  The recursive read-what's-left sees 99, notes it's <= 127, and simply returns it.  The end result is 99*128+2 = 12674.

The file has a header of 'MvG_Passivator', then two sections: the class section and the object section.  The class section starts with an int giving the number of class definitions to follow, then a sequence of class definitions of the form:
	(int)  size of string to follow
	string containing class name
	(int)  class's format integer
	(int)  size of string to follow
	string containing instVarNames

The object section starts with an integer giving the number of objects structures to follow, then a sequence of object structures.  In the object structures are references (denoted refs) to arbitrary objects, described further below in this comment.  The structures are each of the form:
	(int)  class index
	if object class isVariable then
		(int)  object basicSize
	for i=1 to object class instSize
		(ref)  object instVarAt: i
	if object class isBits then
		(bytes)  raw object contents as a series of bytes
	else
		for i=1 to object basicSize
			(ref)  object basicAt: i

The object section has one more thing in it; one ref indicating which object this file represents (usually object #1, but not for primitive objects).

All references (refs) to objects are first converted through the following transformation to get an int (which is written in place of the ref).
	nil -> 0
	true -> 4
	false -> 8
	aCharacter  -> 4n+12,  (where n = aCharacter asInteger)
	aPositiveInteger -> 8n+5
	aNegativeInteger -> -8n+1
	aBehavior -> 4n+2,  (where n = the class's index)
	anObject -> 4n+3   (where n = the object's index)
"

	self error: 'Documentation only'.!
!
```

```
!Passivator methodsFor: 'private-passes'!
collectClasses
	"Figure out which classes need to be included in the output."

	1 to: allRepresentatives size do: [:i |
		| class |
		class := (allRepresentatives at: i) object class.
		(classesToIndex includesKey: class) ifFalse: [
			classesToIndex at: class put: classesToIndex size]].!
collectObjects

	stack := OrderedCollection new: 100.
	self incrementingRepresentativeFor: root.
	[stack size > 0] whileTrue: [
		stack removeLast createSubrepresentativesIn: self].!
!


!Passivator methodsFor: 'private-converting'!
convert: anObject
	"Answer a ByteArray representing the passivation of anObject."

	| result |
	objectToRepresentative := PassivationLargeIdentityDictionary new.
	allRepresentatives := OrderedCollection new.
	classesToIndex := IdentityDictionary new: 50.
	root := anObject.
	self collectObjects.
	self sortObjectsReverseBottomUp.
	self collectClasses.
	bytes := PrivatePassivationByteArray new: (allRepresentatives size * 20 + 50).  "guess size"
	bytesIndex := 0.  "used in a pre-increment way"
	self writeRawBytes: self class headerString.
	self writeClasses.
	self writeObjects.
	result := ByteArray new: bytesIndex.
	result replaceBytesFrom: 1 to: bytesIndex with: bytes startingAt: 1.
	^result!
!


!Passivator methodsFor: 'private-representation'!
incrementingRepresentativeFor: anObject
	"Answers a representative for anObject, or anObject itself if it's special.  Create a
	 representative if necessary, adding it to my stack."

	| rep |
	rep := self representativeFor: anObject.
	rep == anObject ifFalse: [
		rep parentCount: rep parentCount + 1].
	^rep!
!


!Passivator methodsFor: 'private-encoding'!
indexOfClass: aClass
	"Answer the non-negative index of aClass."

	^classesToIndex at: aClass!
!
```

```
!Passivator methodsFor: 'private-representation'!
preSavePassivate: anObject

	^anObject preSavePassivation!
representativeFor: anObject
	"Answers a representative for anObject, or anObject itself if it's special.  Create a
	 representative if necessary, adding it to my stack."

	anObject isSpeciallyPassivatedObject ifTrue: [^anObject].
	^objectToRepresentative
		at: anObject
		ifAbsentPut: [
			| rep |
			rep := PassivationRepresentative new
				parentCount: 0;
				object: (self preSavePassivate: anObject).
			allRepresentatives addLast: rep.
			stack addLast: rep].!
!
```

```
!Passivator methodsFor: 'private-passes'!
sortObjectsReverseBottomUp
	"Sort the objects in reverse bottom-up order (but always with the root first)."
	"This is effectively a treadmill algorithm (a la Henry Baker). It sorts a list in
	place by the topological partial order, automatically breaking cycles when
	necessary.
	Build the list of objects, partitioned by parent count. Keep a separate array
	of partition boundaries. As the single-parent objects are traversed and
	converted to no-parent (and sorted) objects, just fix up each child (make
	sure it occurs after the already-sorted boundary, otherwise we're dealing
	with a victim of a broken cycle."

	| sizes treadmill starts nextSlots toDecrement |
	allRepresentatives size <= 1 ifTrue: [
		"At most one object requiring touch-ups exists. Don't sort."
		allRepresentatives size = 1 ifTrue: [
			allRepresentatives first index: 0].
		^self].
	(objectToRepresentative at: root) parentCount: 0. "Force root to be in zero-parents partition."
	"First, construct the treadmill with each rep containing its index into it."
	sizes := OrderedCollection new: 20.
	1 to: allRepresentatives size do: [:i |
		| parentCountPlusOne |
		parentCountPlusOne := (allRepresentatives at: i) parentCount + 1.
		[parentCountPlusOne > sizes size] whileTrue: [sizes addLast: 0].
		sizes at: parentCountPlusOne put: (sizes at: parentCountPlusOne) + 1].
	treadmill := Array new: sizes sum.
	starts := Array new: sizes size + 2. "convenient boundary condition"
	starts at: 1 put: 1.
	2 to: starts size - 1 do: [:i |
		starts at: i put: (starts at: i - 1) + (sizes at: i - 1)].
	starts at: starts size put: (starts at: starts size - 1).
	nextSlots := starts copy.
	"We now have valid partition information, we just need to place the elements there."
	1 to: allRepresentatives size do: [:i |
		| rep parentCount ind |
		rep := allRepresentatives at: i.
		parentCount := rep parentCount.
		ind := nextSlots at: parentCount + 1.
		nextSlots at: parentCount + 1 put: ind + 1.
		rep index: ind.
		treadmill at: ind put: rep].
"	[(starts copyFrom: 2 to: starts size) = (nextSlots copyFrom: 1 to: nextSlots size - 1)] assert."
"	[treadmill all: [:x | x == (treadmill at: x index)]] assert."
"	[treadmill all: [:x | x index between: (starts at: x parentCount + 1) and: (starts at: x parentCount + 2) - 1]] assert."

	"Make sure the root object is first..."
"	[(objectToRepresentative at: root) index = 1] assert: 'Root object should be only object with no parents'."

	"Here's how to decrement an object's parent count..."
	toDecrement := [:rep |
		"Not a special object or an already-sorted object. Move object to start
		 of partition, then advance partition boundary over it."
		| start |
		start := starts at: rep parentCount + 1.
		start = rep index ifFalse: [
			| temp |
			temp := treadmill at: start.
			treadmill at: start put: rep.
			treadmill at: rep index put: temp.
			temp index: rep index.
			rep index: start].
		"Slide the boundary..."
		starts at: rep parentCount + 1 put: start + 1.
		"and fix rep's parentCount..."
		rep decrementParentCount].

	"Ok, our treadmill is ready. Start chugging objects out the left by moving (starts at: 1) forward."
	[starts first = starts last] whileFalse: [
		starts first = (starts at: 2)
			ifTrue: [
				"We have no more zero-parent objects, so break a cycle. Choose an object
				 with as few parents as possible."
				| victimRep |
```

```
                    victimRep := treadmill at: starts first.  "Not in the zero-refs range, but in first non-empty range thereafter."
                    toDecrement value: victimRep]
                ifFalse: [
                    "We have a zero-parent object."
                    | candidate subreps |
                    candidate := treadmill at: starts first.
                    candidate parentCount = 0 ifFalse: [self error: 'Bug in treadmill'].
                    candidate parentCount: nil.
                    starts at: 1 put: starts first + 1.
                    subreps := candidate subrepresentatives.
                    1 to: subreps size do: [:i |
                        | subrep |
                        subrep := subreps at: i.
                        (subrep isSpeciallyPassivatedObject or: [subrep parentCount == nil]) ifFalse: [
                            toDecrement value: subrep]]]].
"   [starts all: [:x | x = (treadmill size + 1)]] assert."

    "Now make sure the algorithm at least produced a meaningful result."
"   [treadmill all: [:x | x == (treadmill at: x index)]] assert."

    "Offset the indices now to be zero-relative, and we're done..."
    1 to: treadmill size do: [:i |
        | r |
        r := treadmill at: i.
        r index: r index - 1].!
!


!Passivator methodsFor: 'private-writing'!
writeClassDescription: class
    "Write a description of the class suitable for version verification.  See whatIsFileFormat on the class side."

    | vars varString |
    self writeSizedString: class fullName.
    vars := class allInstVarNames.
    varString := WriteStream on: (String new: 10 * vars size).
    vars do: [:var | varString nextPutAll: var; space].
    self writeInt: class format.
    self writeSizedString: varString contents.!
!


!Passivator methodsFor: 'private-passes'!
writeClasses

    | array |
    array := Array new: classesToIndex size.
    classesToIndex keysAndValuesDo: [:class :index |
        array at: index + 1 put: class].
    self writeInt: array size.
    array do: [:class |
        self writeClassDescription: class].!
!


!Passivator methodsFor: 'private-writing'!
writeInt: int
    "Write the non-negative integer int to the stream.  See whatIsFileFormat on the class side."

    | remainder |
    remainder := int.
    [remainder > 127] whileTrue: [
        bytes at: (bytesIndex := bytesIndex + 1) put: (remainder bitAnd: 127) + 128.
        remainder := remainder bitShift: -7].
    bytes at: (bytesIndex := bytesIndex + 1) put: remainder.!
!
```

```
!Passivator methodsFor: 'private-passes'!
writeObjects

    | array |
    array := Array new: allRepresentatives size.
    1 to: allRepresentatives size do: [:i |
        | rep |
        rep := allRepresentatives at: i.
        array at: rep index + 1 put: rep].
    self writeInt: array size.
    1 to: array size do: [:i | (array at: i) writePart1For: self].
    1 to: array size do: [:i | (array at: i) writePart2For: self].
    self writeReferenceToRepresentative: (self representativeFor: root).  "indicate which object all this data represents."!
!


!Passivator methodsFor: 'private-writing'!
writeRawBytes: bytesObject
    "Write the object's raw byte data to the stream."

    bytesObject class isBits ifFalse: [self error: 'Object must contain bytes'].
    bytes
        replaceBytesFrom: bytesIndex + 1
        to: (bytesIndex := bytesIndex + bytesObject basicSize)
        with: bytesObject
        startingAt: 1.!
writeReferenceToRepresentative: representativeOrSpecial
    "Write data representing a reference to the object in the representative."

    self writeInt: (representativeOrSpecial passivationIntegerEncodingIn: self)!
writeSizedString: aString
    "Write a string prefixed by its size."

    aString isString ifFalse: [self error: 'Must be a string'].
    self writeInt: aString size.
    self writeRawBytes: aString.!
!

Smalltalk defineClass: #ContinuousPassivator
    superclass: #{Passivator}
    indexedType: #none
    private: false
    instanceVariableNames: ''
    classInstanceVariableNames: ''
    imports: ''
    category: ''!


!ContinuousPassivator class methodsFor: 'accessing'!
headerString
    "Answer the prefix used to identify this format."

    ^'CP5i'!
!
```

```
!ContinuousPassivator methodsFor: 'private-converting'!
convert: anObject
	"Answer a ByteArray representing the passivation of anObject."

	| firstClass result |
	classesToIndex isNil ifTrue: [
		"Only initialize if this instance hasn't been used yet."
		classesToIndex := IdentityDictionary new].
	firstClass := classesToIndex size.
	objectToRepresentative := PassivationLargeIdentityDictionary new.
	allRepresentatives := OrderedCollection new.
	root := anObject.
	self collectObjects.
	self sortObjectsReverseBottomUp.
	self collectClasses.
	bytes isNil ifTrue: [bytes := PrivatePassivationByteArray new: (allRepresentatives size * 20 + 50)]. "guess size"
	bytesIndex := 0. "used in a pre-increment way"
	self writeRawBytes: self class headerString.
	self writeClassesStartingAt: firstClass.
	self writeObjects.
	objectToRepresentative := root := stream := nil. "save some memory"
	result := ByteArray new: bytesIndex.
	result replaceBytesFrom: 1 to: bytesIndex with: bytes startingAt: 1.
	^result!
!


!ContinuousPassivator methodsFor: 'private-passes'!
writeClassesStartingAt: firstClass
	"The only difference between ContinuousPassivator and Passivator
	 is that this method doesn't write out class definitions it has previously
	 encoded.  The count is still the same, but the initial N class definitions
	 are simply omitted."

	| array |
	array := Array new: classesToIndex size.
	classesToIndex keysAndValuesDo: [:class :index |
		array at: index + 1 put: class].
	self writeInt: array size.
	firstClass + 1 to: array size do: [:arrayIndex |
		self writeClassDescription: (array at: arrayIndex)].!
!

"-------------------------------------------------------------------------"
Smalltalk defineClass: #MiosoftPassivator
	superclass: #{ContinuousPassivator}
	indexedType: #none
	private: false
	instanceVariableNames: ''
	classInstanceVariableNames: ''
	imports: ''
	category: ''!


!MiosoftPassivator class methodsFor: 'accessing'!
headerString
	"Answer the prefix used to identify this format."

	^'M'  "M is for Miosoft."!
!


!MiosoftPassivator methodsFor: 'private-representation'!
preSavePassivate: anObject

	anObject ooIsPersistent
		ifTrue: [^anObject preSavePassivationForPersistentObject]
		ifFalse: [^anObject preSavePassivation].!
!
```

```
Smalltalk.Applications defineClass: #MarshallingFramework
	superclass: #{ENVY.Application}
	indexedType: #none
	private: false
	instanceVariableNames: ''
	classInstanceVariableNames: ''
	imports: ''
	category: 'ENVY/Manager'!


!Core.Behavior methodsFor: ':Passivation fixups'!
isSpeciallyPassivatedObject

	^true!
passivationIntegerEncodingIn: aPassivator

	^(aPassivator indexOfClass: self) * 4 + 2!
!


!Core.Character methodsFor: ':Passivation fixups'!
isSpeciallyPassivatedObject

	^true!
passivationIntegerEncodingIn: aPassivator

	^self asInteger * 4 + 12!
!


!Kernel.CompiledCode methodsFor: ':Passivation passivation/activation'!
preSavePassivation
	"I should not be passivated.  Report an error if an attempt is made."

	self error: 'Should not passivate Compiled Code'.!
!


!Core.False methodsFor: ':Passivation fixups'!
isSpeciallyPassivatedObject

	^true!
passivationIntegerEncodingIn: aPassivator

	^8!
!


!Graphics.GraphicsHandle methodsFor: ':Passivation passivation/activation'!
preSavePassivation
	"I should not be passivated.  Report an error if an attempt is made."

	self error: 'Should not passivate GraphicsHandle'.!
!


!Core.Integer methodsFor: ':Passivation fixups'!
isSpeciallyPassivatedObject

	^true!
passivationIntegerEncodingIn: aPassivator

	^self < 0
		ifTrue: [self * -8 + 1]
		ifFalse: [self * 8 + 5]!
!
```

```
!Core.Object class methodsFor: ':Passivation fixups'!
activatorFixupForFormat: formatInt instVars: fileInstVars isVariable: isVariable isBits: isBits
	"Answer a block which, when evaluated at the right time with an instance
	 of me and an Activator, will initialize the instance.  Make sure to read the
	 right amount of data from the activator."

	| imageInstVars map |
	imageInstVars := self allInstVarNames.
	map := fileInstVars collect: [:str | imageInstVars indexOf: str ifAbsent: [nil]].

	^[:object :activator |
		"Read and set the inst vars first..."
		map do: [:destInstVar |
			| val |
			val := activator objectFromIndex: activator readInt.
			destInstVar notNil ifTrue: [object instVarAt: destInstVar put: val]].
		"Now check for variable / bits data..."
		isVariable ifTrue: [
			isBits
				ifTrue: [activator readRawBytesInto: object]
				ifFalse: [
					1 to: object basicSize do: [:i |
						object basicAt: i put: (activator objectFromIndex: activator readInt)]]]].!
!


!Core.Object methodsFor: ':Passivation passivation/activation'!
isSpeciallyPassivatedObject

	^false!
passivationIntegerEncodingIn: aPassivator

	self error: 'Must overide for specially passivated objects'.!
postLoadActivation: anActivator
	"Activate myself now that I have been loaded by anActivator.  Answer the object
	 to substitute for myself in parent objects (when not involved in cycles).  Subclasses
	 should reimplement if there is a need to translate the object in some way during activation."

	^self!
preSavePassivation
	"Answer an object to passivate in place of myself.  Subclasses should reimplement
	 if there is a need to translate the object in some way during passivation."

	^self!
preSavePassivationForPersistentObject
	"Answer an object to passivate in place of myself.  Since I must be a persistent object, passivate
	 a special oid-like object in place of me."

	self ooIsPersistent ifFalse: [self error: 'This object must be persistent'].
	^PassivatedPersistentObjectReference new oid: self oid!
!


!OoStub methodsFor: 'passivation'!
preSavePassivationForPersistentObject
	"Answer an object to passivate in place of myself.  Since I must be a persistent object, passivate
	 a special oid-like object in place of me."

	^PassivatedPersistentObjectReference new oid: self oid!
!


!OoVArray methodsFor: 'passivation/activation'!
preSavePassivationForPersistentObject
	"Answer an object to passivate in place of myself.  Even though I'm persistent, I don't have an oid of
	 my own (that's just how the Objectivity/DB works).  Passivate a transient copy of me instead."

	self ooIsPersistent ifFalse: [self error: 'This object must be persistent'].
	^self copy!
!
```

```
!Core.Set methodsFor: ':Passivation passivation/activation'!
postLoadActivation: anActivator
	"Activate myself now that I have been loaded by anActivator.  Answer the object
	 to substitute for myself, which is simply myself after rehashing my elements."

	self rehash.
	^self!
!


!Core.SmallInteger methodsFor: ':Passivation fixups'!
passivationIntegerEncodingIn: aPassivator

	^self < 0
		ifTrue: [self * -8 + 1]
		ifFalse: [self * 8 + 5]!
!


!Core.Symbol methodsFor: ':Passivation passivation/activation'!
postLoadActivation: anActivator
	"Activate myself now that I have been loaded by anActivator.  Answer the object
	 to substitute for myself, which in this case is the interned version of me."

	^self class intern: self!
!


!Core.True methodsFor: ':Passivation fixups'!
isSpeciallyPassivatedObject

	^true!
passivationIntegerEncodingIn: aPassivator

	^4!
!


!Core.UndefinedObject methodsFor: ':Passivation fixups'!
isSpeciallyPassivatedObject

	^true!
passivationIntegerEncodingIn: aPassivator

	^0!
!
```